

Rapport projet : Bot discord

Lien GITHUB : <https://github.com/YannisVa/BotDiscordM1SDTSFA>

Lien vidéo : <https://www.youtube.com/watch?v=yB6iF3TiuY>

Table des matières :

1 QU'EST-CE QU'UN BOT ?	2
2 CONTEXTE DU PROJET :	2
3 PRESENTATION SUCCINCTE DU PROJET :	2
4 CONTRAINTES DONNEES POUR LE PROJET :	2
5 INSTALLATION ET PREPARATION DU BOT DISCORD :	3
5.1 CREATION DU BOT :	3
5.2 INSTALLATION DE PYHTON 3.9.7 :	8
5.3 INSTALLATION NLTK :	10
5.4 EXECUTION DU PROGRAMME :	11
6 FONCTIONNALITES DU BOT DISCORD :	12
7 EXPLICATION DES DIFFERENTS FICHIERS NECESSAIRES POUR CE PROJET :	17
7.1 PROJETBOTDISCORD.PY :	17
7.2 INTENTS.JSON :	24
8 CONCLUSION :	25

1 Qu'est-ce qu'un bot ?

Un bot est un système programmé pour réaliser de multiples tâches automatisées. En effet, les bots ont principalement comme but de réaliser des séries tâches automatiquement lorsqu'un évènement apparaît. Un évènement peut être un message provenant d'une source externe, un horaire spécifique ou bien une action sur machine particulière. Le but des bots va être d'imiter les comportements humains en effectuant des tâches répétitives plus rapidement qu'un humain.

Parmi les bots existants actuellement, les deux types de bots les plus connus sont :

Chatbot : Les bots de ce type ont pour but de mener une conversation avec un utilisateur en utilisant uniquement des messages de type texte. Les utilisateurs et les bots de ce type peuvent alors communiquer entre eux à l'écrit. Exemple : ChatBot Facebook

Voicebot : Les bots de ce type ont pour but de mener des conversations avec les utilisateurs en communiquant vocalement. C'est à dire que l'utilisateur peut parler directement aux bots de ce type sans que des messages textuels soient envoyés. Exemple : Alexa ou Google Home

Exemple de bot que l'on utilise quotidiennement :

Les bots sont actuellement présents dans notre vie de tous les jours. Pour les adeptes des réseaux sociaux comme Facebook, Messenger, Telegram ou bien même Instagram, nous retrouvons des bots dotés de système interactifs robotisés pour répondre à des demandes ou bien même juste pour communiquer avec des utilisateurs. Le but principal est donc de « tchatter » avec un robot qui peut nous guider vers les actions que nous souhaitons faire. Le fait de « tchatter » avec un robot permet d'avoir un comportement plus ou moins humain avec l'utilisateur.

Les bots existent aussi dans les jeux vidéo par exemple. Certains éditeurs de jeux utilisent des personnages non joueurs dotés d'IA permettant au joueur de discuter avec ce dernier pour le guider dans son aventure.

Les bots existent aussi dans le domaine médical où nous retrouvons des bots du type chatbot permettant à l'utilisateur de diagnostiquer ces maladies à l'aide d'une conversation avec un robot. Le bot est alors représenté comme un agent virtuel, il est directement en relation avec les clients et travaille en autonomie pour aider les différents utilisateurs à comprendre leurs symptômes.

2 Contexte du projet :

Dans le cadre de ma formation en master 1 OIVM parcours Système distribué et data science et plus précisément dans le cadre du cours : conception et analyse d'algorithmes (CAA), j'ai pour but de réaliser un bot en utilisant l'API (application programming interface) discord.py.

3 Présentation succincte du projet :

Ce projet a pour but de créer un bot discord en utilisant l'API discord.py. Le bot créé doit donc être capable de réaliser des tâches d'administration de serveurs mais il doit aussi pouvoir communiquer avec les utilisateurs et répondre à un évènement en effectuant une série de tâches.

4 Contraintes données pour le projet :

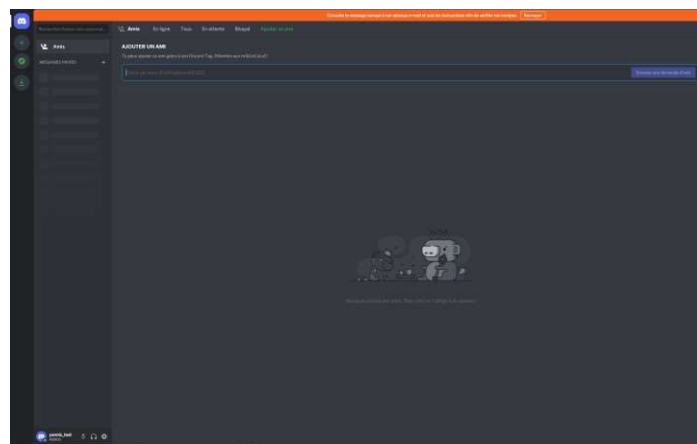
- Utilisation du langage de programmation python ou C++
- Utilisation de l'API discord.py (**ATTENTION** : Discord prévoit la fin de discord.py pour **avril 2022** : <https://www.lemondedupc.fr/article/92-la-fin-de-discord-py>)

5 Installation et préparation du Bot discord :

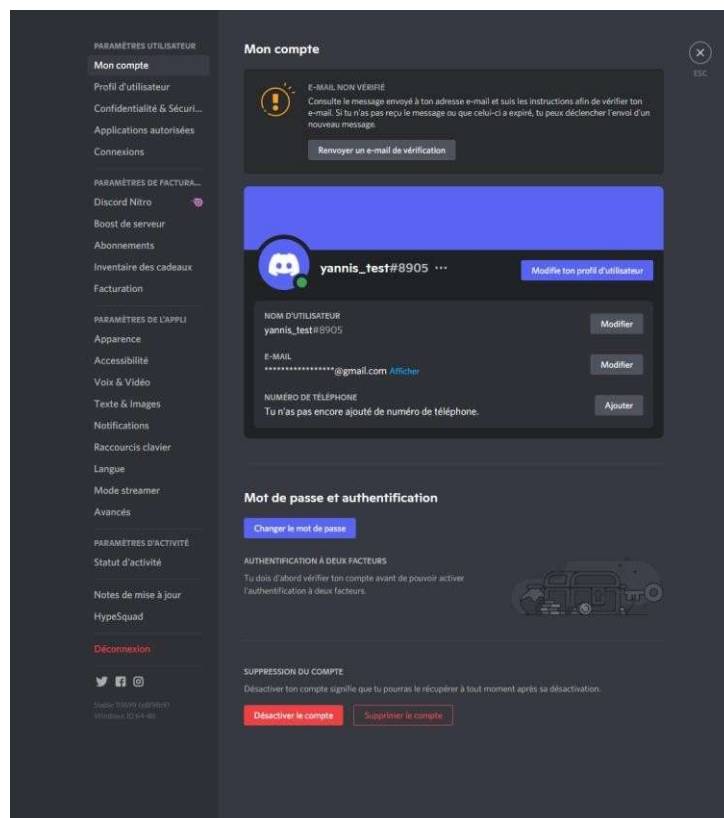
5.1 Création du bot :

Pour créer le Bot il est nécessaire d'activer la partie développeur de son compte discord. Cette option est activable dans les paramètres de discord. Il est préférable d'utiliser discord sur le navigateur pour activer cette option : <https://discord.com/>

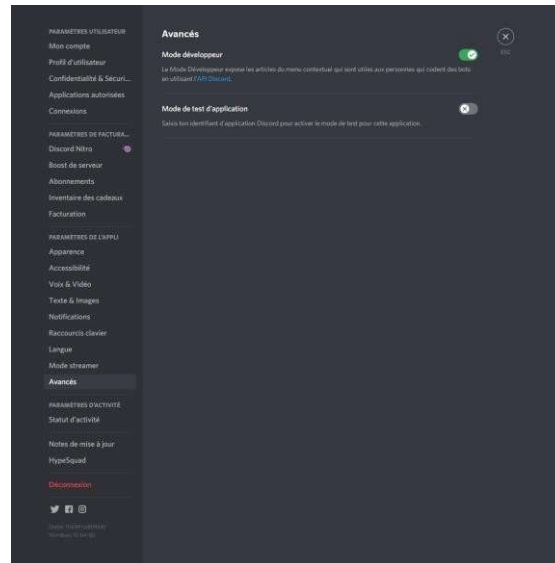
Une fois connecté à discord sur le navigateur, il faut cliquer sur l'engrenage pour accéder aux paramètres :



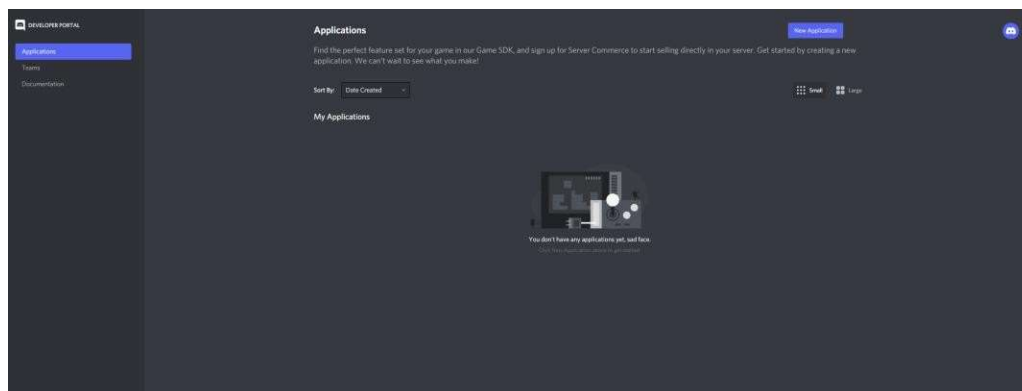
Vous arrivez ensuite sur une page comme celle-ci :



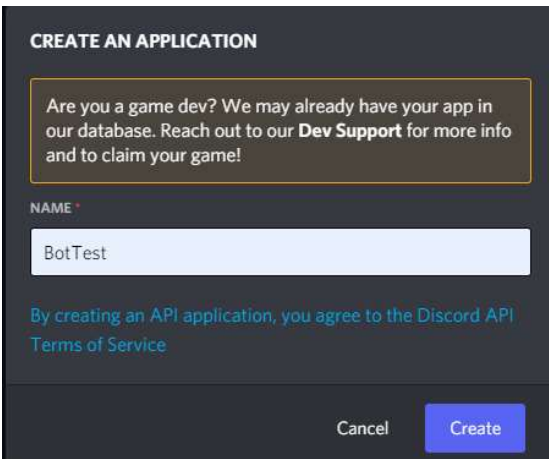
Sur cette page, il faudra se rendre dans la rubrique « Avancés » pour cocher le mode développeur :



Une fois cela fait, il faudra ouvrir un nouvel onglet pour se rendre sur la page développeur de discord (Voici le lien : <https://discord.com/developers/applications>) :



Sur cette page il sera nécessaire de cliquer sur « New application ». Une fenêtre apparaîtra vous demandant de renseigner un nom pour l'application :



CREATE AN APPLICATION

Are you a game dev? We may already have your app in our database. Reach out to our **Dev Support** for more info and to claim your game!

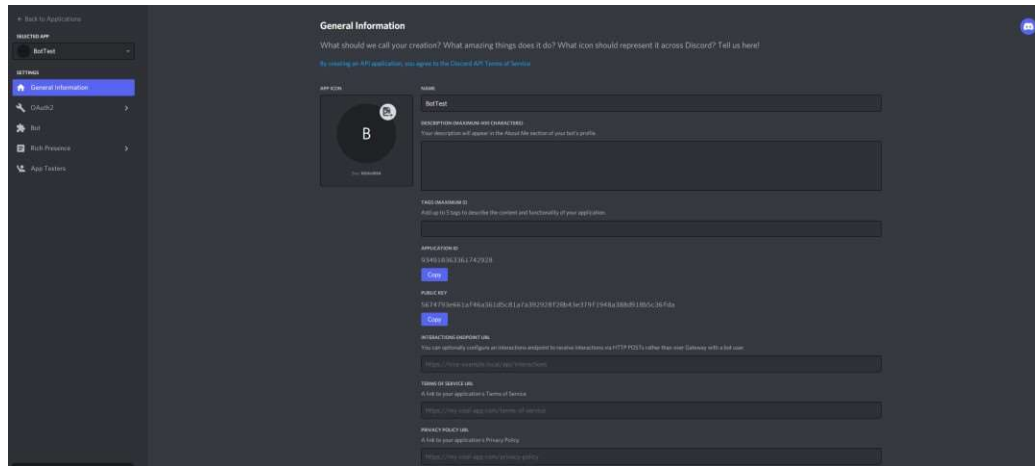
NAME *

BotTest

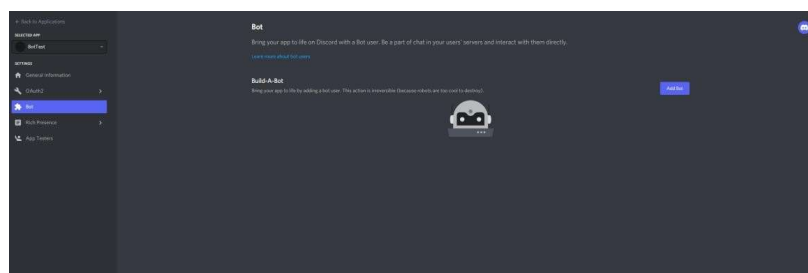
By creating an API application, you agree to the [Discord API Terms of Service](#)

Cancel Create

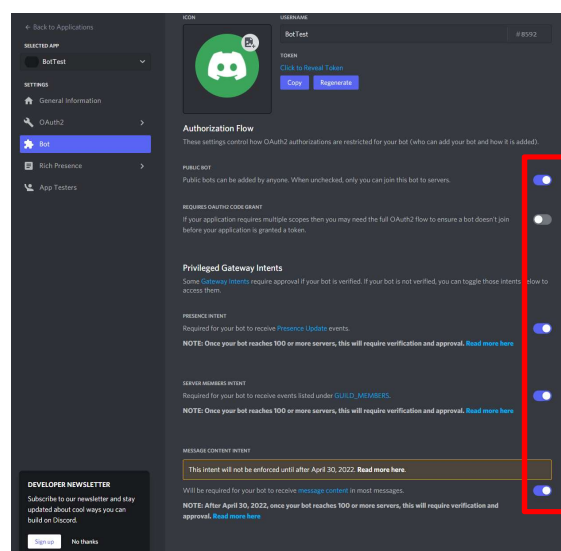
Après avoir cliqué sur « Create » vous arriverez sur cette page. La partie Bot est la partie qui nous intéresse pour mettre en marche le bot.



En cliquant sur la rubrique bot vous pourrez ensuite cliquer sur le bouton « Add Bot » :



Après avoir cliqué sur ce bouton vous arriverez sur une page dans laquelle vous pouvez personnaliser votre robot discord. Cocher les options ci-dessous :



Récupérer le TOKEN du bot en cliquant sur « Click to reveal Token » puis sur « copy ». Ce token est nécessaire pour lancer le bot. Il est donc nécessaire d'avoir cette information avant de lancer le projet.py.

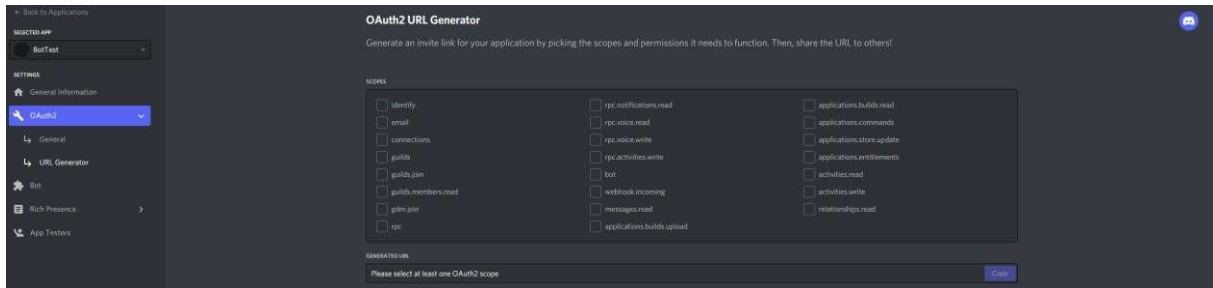
Remarque : Il ne faut pas oublier de cocher les options se trouvant dans le rectangle rouge ci-dessus.

Etudiant : Vaulry Yannis

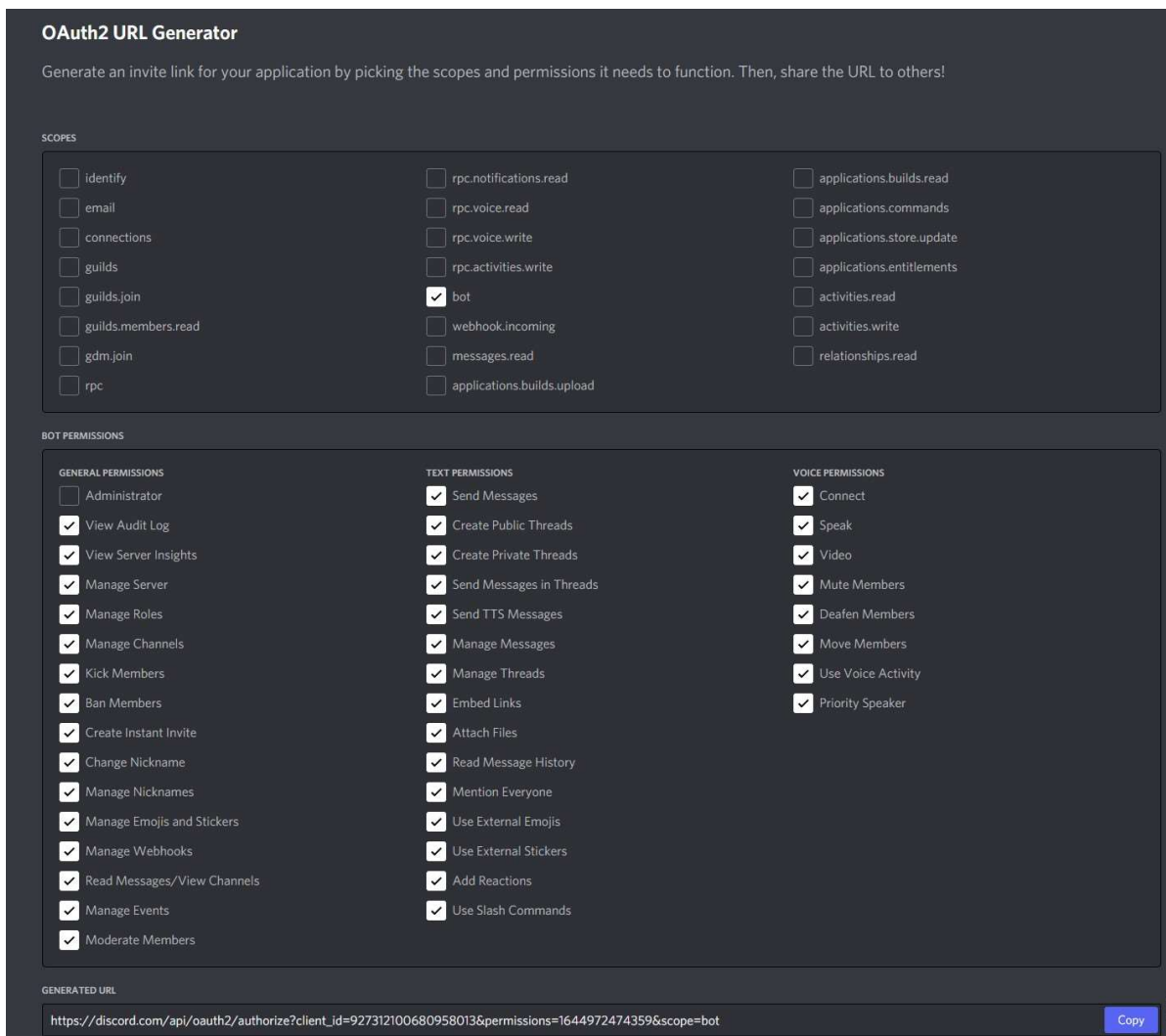
Master : SDTS FA

Rapport créé le : 14/02/2022

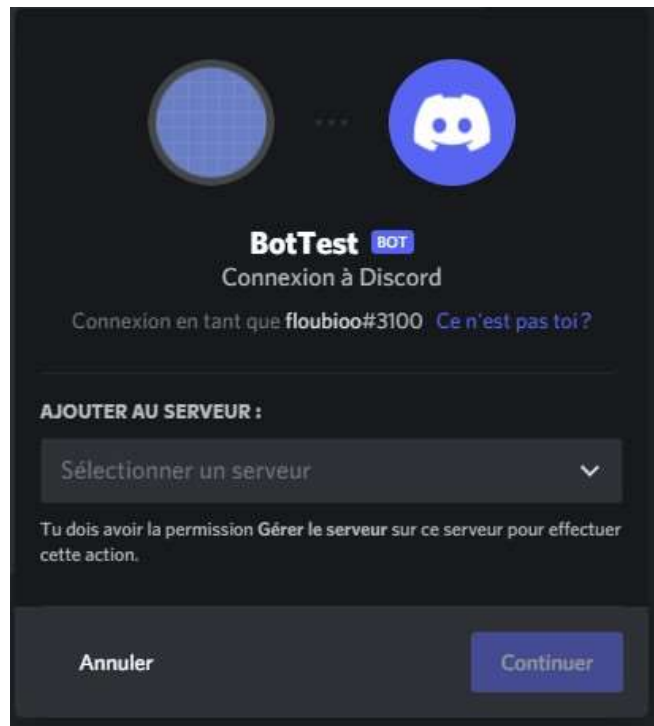
Une fois cela fait, vous devez cliquer sur la rubrique OAuth2 pour ensuite sélectionner URL Generator :



Une fois cela fait vous devrez cocher le champ « bot » et cocher les champs ci-dessous puis vous devez cliquer sur le bouton « copy » pour copier le lien permettant d'ajouter le bot à un serveur discord.



Sur le lien vous pourrez spécifier le serveur sur lequel le bot sera actif (si le lien n'a pas fonctionné, vous devez vérifier que vous êtes bien connecté sur discord dans un autre onglet) :



Une fois cela fait, le bot sera présent dans votre serveur discord. On peut voir que le bot est inactif (hors ligne).



5.2 Installation de Python 3.9.7 :

L'installation de python et de ses différentes librairies nécessitent d'avoir un espace mémoire disponible supérieur à 1 go. Maintenant que le bot a été créé il faut installer python 3.9.7. Sur Windows vous pouvez télécharger et installer cette version de python via le lien : <https://www.python.org/downloads/release/python-397/>

Lors de l'installation **Windows**, vérifier que vous avez ajouté python 3.9 to PATH :



Il faudra ensuite mettre à jour pip qui nous permettra de télécharger les différentes librairies. Pour le mettre à jour, il faudra ouvrir une fenêtre CMD et renseigner cette commande :

`py -m pip install --upgrade pip`

```
C:\Users\YannisPCB>py -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (21.3.1)
```

Une fois cela fait il sera possible d'installer les différentes librairies :

`discord.py: py -m pip install discord.py`

```
C:\Users\YannisPCB>py -m pip install discord.py
Requirement already satisfied: discord.py in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (1.7.3)
Requirement already satisfied: aiohttp<3.8.0,>=3.6.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from discord.py) (3.7.4.post0)
Requirement already satisfied: attrs<=17.3.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (21.4.0)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (5.2.0)
Requirement already satisfied: charset<5.0,>=2.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (4.0.0)
Requirement already satisfied: typing-extensions<=3.6.5 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (3.7.4.3)
Requirement already satisfied: async-timeout<4.0,>=3.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (3.0.1)
Requirement already satisfied: yarl<2.0,>=1.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp<3.8.0,>=3.6.0->discord.py) (1.7.2)
Requirement already satisfied: idna<=2.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from yarl<2.0,>=1.0->aiohttp<3.8.0,>=3.6.0->discord.py) (3.3)
```

`deepl : py -m pip install deepl`

```
C:\Users\YannisPCB>py -m pip install deepl
Requirement already satisfied: deepl in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (1.3.1)
Requirement already satisfied: requests<3,>=2 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from deepl) (2.27.0)
Requirement already satisfied: idna<4,>=2.5 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from requests<3,>=2->deepl) (3.3)
Requirement already satisfied: charset-normalizer<=2.0.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from requests<3,>=2->deepl) (2.0.10)
Requirement already satisfied: certifi<=2017.4.17 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from requests<3,>=2->deepl) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from requests<3,>=2->deepl) (1.26.7)
C:\Users\YannisPCB>
```

Etudiant : Vaulry Yannick

Master : SDTS FA

Rapport créé le : 14/02/2022

Neuralintents : py -m pip install neuralintents

```
C:\Users\YannisPC>py -m pip install neuralintents
Requirement already satisfied: neuralintents in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (0.0.4)
Requirement already satisfied: tensorflow in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from neuralintents) (2.5.0)
Requirement already satisfied: nltk in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from neuralintents) (3.6.7)
Requirement already satisfied: numpy in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from neuralintents) (1.19.5)
Requirement already satisfied: regex<=2021.8.3 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from nltk->neuralintents) (2021.11.10)
Requirement already satisfied: joblib in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from nltk->neuralintents) (1.1.0)
Requirement already satisfied: click in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from nltk->neuralintents) (8.0.3)
Requirement already satisfied: tqdm in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from nltk->neuralintents) (4.62.3)
Requirement already satisfied: termcolor<=1.1.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from tensorflow->neuralintents) (1.1.0)
Requirement already satisfied: grpcio<=1.34.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from tensorflow->neuralintents) (1.34.1)
```

Discord-slash : py -m pip install discord-py-slash-command (ou "py -m pip install -U discord-py-slash-command")

```
C:\Users\YannisPC>py -m pip install discord-py-slash-command
Requirement already satisfied: discord-py-slash-command in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (3.0.3)
Requirement already satisfied: aiohttp in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from discord-py-slash-command) (3.7.4.post0)
Requirement already satisfied: discord.py in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from discord-py-slash-command) (1.7.3)
Requirement already satisfied: async-timeout<4.0,>=3.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (3.0.1)
Requirement already satisfied: chardet<5.0,>=2.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (4.0.0)
Requirement already satisfied: typing-extensions<=3.6.5 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (3.7.4.3)
Requirement already satisfied: yarl<2.0,>=1.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (1.7.2)
Requirement already satisfied: attrs<=17.3.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (21.4.0)
Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from aiohttp->discord-py-slash-command) (5.2.0)
Requirement already satisfied: idna<=2.0 in c:\users\yannispcb\appdata\local\programs\python\python39\lib\site-packages (from yarl<2.0,>=1.0->aiohttp->discord-py-slash-command) (3.3)
```

Pour **Debian 10 ou 11** il faudra réaliser les 16 commandes ci-dessous dans un terminal :

1. sudo apt update
2. sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libsqlite3-dev libreadline-dev libffi-dev curl libbz2-dev
3. wget <https://www.python.org/ftp/python/3.9.7/Python-3.9.7.tgz>
4. tar -xf Python-3.9.7.tgz
5. cd Python-3.9.7
6. ./configure --enable-optimizations
7. make -j 2
8. sudo make altinstall
9. python3.9 --version
10. sudo apt install libffi-dev libnacl-dev python3-dev
11. sudo apt update
12. sudo apt install python3-pip
13. python3 -m pip install -U discord.py
14. python3 -m pip install -U deep
15. python3 -m pip install -U neuralintents
16. python3 -m pip install -U discord-py-slash-command

Concernant **MAC OS**, il faudra télécharger python 3.9.7 sur :
<https://www.python.org/downloads/release/python-397/>

Une fois cela fait, il faudra réaliser les commandes ci-dessous dans un terminal pour installer pip (Sur certaines versions de MAC OS, il est possible d'utiliser les commandes 12 à 16 de la partie Debian) :

1. `curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py`
2. `python3 get-pip.py`
3. `pip install discord.py`
4. `pip install deepl`
5. `pip install neuralintents`
6. `pip install discord-py-slash-command`

5.3 Installation NLTK :

Une fois les librairies importées, il est nécessaire de terminer l'installation de nltk pour le fonctionnement de neuralintents. Nous devons tout d'abord accéder à la console python.

- Sur **Windows**, il faut renseigner py dans un terminal pour accéder à la console Python.
- Sur **Linux** et **MAC OS**, il est possible d'utiliser la commande `python3` dans un terminal.

Lorsque vous êtes dans la console python, vous devez faire ceci :

```
>>> import nltk  
>>> nltk.download('omw-1.4')
```

Cela permet de télécharger nltk et d'éviter le message d'erreur ci-dessous lors de l'exécution du projet.py :

```
Resource omw-1.4 not found.  
Please use the NLTK Downloader to obtain the resource:
```

```
>>> import nltk  
>>> nltk.download('omw-1.4')
```

For more information see: <https://www.nltk.org/data.html>

```
Attempted to load corpora/omw-1.4.zip/omw-1.4/
```

```
Searched in:  
- '/root/nltk_data'  
- '/usr/nltk_data'  
- '/usr/share/nltk_data'  
- '/usr/lib/nltk_data'  
- '/usr/share/nltk_data'  
- '/usr/local/share/nltk_data'  
- '/usr/lib/nltk_data'  
- '/usr/local/lib/nltk_data'
```

5.4 Exécution du programme :

Pour l'exécution du projet.py sous Windows, il faudra placer dans un même répertoire le fichier intents.json et le fichier ProjetBotDiscord.py. Une fois cela fait, il faudra ouvrir un CMD pour faire un cd à l'emplacement du répertoire et lancer la commande py ProjetBotDiscord.py sur Windows et python3 ProjetBotDiscord.py sur Debian et MAC OS.

Le programme se lancera et chargera le modèle intents.json. Une fois l'entraînement terminé du bot, le programme vous demandera le Token récemment récupéré :

```
Invite de commandes - py ProjetBotDiscord.py
10/10 [=====] - 0s 441us/step - loss: 1.2134 - accuracy: 0.4130
Epoch 187/200
10/10 [=====] - 0s 551us/step - loss: 1.2612 - accuracy: 0.4130
Epoch 188/200
10/10 [=====] - 0s 441us/step - loss: 1.1948 - accuracy: 0.4348
Epoch 189/200
10/10 [=====] - 0s 496us/step - loss: 1.1976 - accuracy: 0.4348
Epoch 190/200
10/10 [=====] - 0s 496us/step - loss: 1.1804 - accuracy: 0.4348
Epoch 191/200
10/10 [=====] - 0s 496us/step - loss: 1.1922 - accuracy: 0.4565
Epoch 192/200
10/10 [=====] - 0s 441us/step - loss: 1.2041 - accuracy: 0.4783
Epoch 193/200
10/10 [=====] - 0s 441us/step - loss: 1.1911 - accuracy: 0.5000
Epoch 194/200
10/10 [=====] - 0s 441us/step - loss: 1.2318 - accuracy: 0.4565
Epoch 195/200
10/10 [=====] - 0s 496us/step - loss: 1.2029 - accuracy: 0.4565
Epoch 196/200
10/10 [=====] - 0s 496us/step - loss: 1.1790 - accuracy: 0.4348
Epoch 197/200
10/10 [=====] - 0s 441us/step - loss: 1.2065 - accuracy: 0.4783
Epoch 198/200
10/10 [=====] - 0s 551us/step - loss: 1.1820 - accuracy: 0.4565
Epoch 199/200
10/10 [=====] - 0s 496us/step - loss: 1.2085 - accuracy: 0.3913
Epoch 200/200
10/10 [=====] - 0s 441us/step - loss: 1.1951 - accuracy: 0.4783
Veuillez renseigner l'id (TOKEN du bot) :
```

Le programme vous demandera ensuite d'ajouter les différents utilisateurs admin.

La touche « 0 » permet de quitter la boucle permettant d'ajouter les profils admin.

Lorsque le programme affichera « Bot connecté » et « Bot prêt », vous pourrez l'utiliser (il sera alors visible dans « état vert » dans discord :

```
Veuillez renseigner l'id (TOKEN du bot) : OTM00TEwMzYzMzYxNzQyOTI4.Ye29IA.d97kG-a8QhSEuij0LJcrrhbVsdU
Qui peut avoir le droit admin (ex : TOTO#212)
Pour quitter utiliser le chiffre '0'
Renseigner l'identifiant : floubioo#3100
['floubioo#3100']
Qui peut avoir le droit admin (ex : TOTO#212)
Pour quitter utiliser le chiffre '0'
Renseigner l'identifiant : 0
Bot connecté
Bot prêt
```

Remarque : Si l'installation de neuralintents « tensorflow » via pip n'a pas fonctionné, il est alors possible d'utiliser le fichier ProjetBotDiscordSNI.py. Ce fichier comporte les mêmes fonctionnalités que le fichier par défaut mais il ne contient pas neuralintents. La fonctionnalité « chat » du bot ne sera donc pas possible.

6 Fonctionnalité du bot discord :

Lorsque le bot est connecté sur discord et qu'il est possible de communiquer avec lui avec les différentes commandes ci-dessous (ces commandes sont visibles en utilisant la commande **!aide**) :

Parmi toutes ces commandes, il est nécessaire dans les expressions d'éviter d'utiliser des caractères spéciaux. Certaines de ces fonctions ne les prennent pas en charge. Le bot ne renverra donc pas de réponse à l'utilisateur.

Commande : **!dire [expression]**

Description : Cette commande permet de faire répéter un mot ou une phrase au bot que vous avez créé.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!traduireEN [expression]**

Description : Cette commande demande au bot de traduire en anglais un mot ou une phrase. Pour cela le bot utilise directement la librairie « deepl » de python. Dans le code un TOKEN a été renseigné pour effectuer cela.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!traduireFR [expression]**

Description : Cette commande demande au bot de traduire en français un mot ou une phrase. Pour cela le bot utilise directement la librairie « deepl » de python. Dans le code un TOKEN a été renseigné pour effectuer cela.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!traduireES[expression]**

Etudiant : Vaulry Yannis

Master : SDTS FA

Rapport créé le : 14/02/2022

Description : Cette commande demande au bot de traduire en espagnol un mot ou une phrase. Pour cela le bot utilise directement la librairie « deepL » de python. Dans le code un TOKEN a été renseigné pour effectuer cela.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!traduireDE[expression]**

Description : Cette commande demande au bot de traduire en allemand un mot ou une phrase. Pour cela le bot utilise directement la librairie « deepL » de python. Dans le code un TOKEN a été renseigné pour effectuer cela.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!transport [transport] [nom du transport] [position de départ]**

Description : Cette commande demande au bot de vous donner l'horaire correspondant au départ d'un transport pour aller dans une direction donnée. Pour cela le bot envoie une requête http à l'API de la RATP (<https://api-ratp.pierre-grimaud.fr/v4>).

Parmi les paramètres à transmettre au bot vous devez spécifier :

1. [transport] -> Correspond au type de transport que vous souhaitez emprunter. Parmi les transports pris en charge il y'a :

- buses -> bus
- metros -> métro
- tramways -> tramway

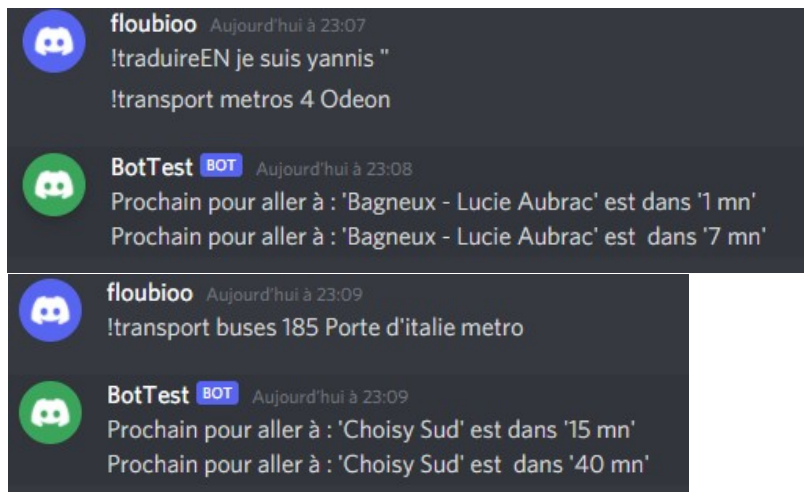
2. [nom du transport] -> Correspond au nom du transport exemple : 4 pour accéder au métro 4 ou bien 185 pour accéder au bus 185...

3. [positions de départ] -> Correspond à votre point de départ, exemple : Mairie de Montrouge pour le métro 4 ou bien Choisy Sud pour le bus 185. Pour obtenir le nom complet des différentes station vous pouvez utiliser la fonction du bot : **!stationsT [transport] [nom du transport]**.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Elle renvoie aucune information lorsque les transports sont à l'état : « Service terminé »

Exemple :



Commande : **!chat [expression]**

Description : Cette commande permet de discuter avec le bot sans que ce dernier réalise des commandes.

Contrainte(s) : Cette commande ne prend pas en compte le caractère '«'.

Exemple :



Commande : **!calculatrice [expression]**

Description : Cette commande permet d'afficher graphiquement une calculatrice pour effectuer des calculs simplistes.

Contrainte(s) : Cette commande nécessite d'être terminée via le bouton « fin ». Cette fonction fonctionne uniquement en message privé (pour ne pas qu'un utilisateur du serveur utilise votre calculatrice).

Exemple :



Commande : **!genererP [nbr de caractères]**

Description : Cette commande permet de générer un mot de passe aléatoire comprenant des chiffres, des majuscules, des minuscules et des caractères spéciaux.

Contrainte(s) : Cette fonction nécessite d'avoir en paramètre un nombre ≥ 6 . Cette fonction fonctionne uniquement en message privé (pour ne pas qu'un utilisateur du serveur voit votre mot de passe).

Exemple :



Commande : **!creerCV [nom]**

Description : Cette commande permet de créer un salon vocal.

Contrainte(s) : Cette fonction est utilisable uniquement pour les personnes dites « admin ». Cette fonction ne fonctionne pas en message privé.

Exemple :



Commande : **!creerCT [nom]**

Description : Cette commande permet de créer un salon textuel.

Contrainte(s) : Cette fonction est utilisable uniquement pour les personnes dites « admin ». Cette fonction ne fonctionne pas en message privé.

Exemple :

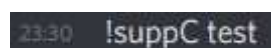


Commande : **!suppC [nom]**

Description : Cette commande permet de supprimer un salon.

Contrainte(s) : Cette fonction est utilisable uniquement pour les personnes dites « admin ». Cette fonction ne fonctionne pas en message privé.

Exemple :



Commande : **!ban** [**@toto**] [**raison du ban**]

Description : Cette commande permet de bannir une personne du serveur.

Contrainte(s) : Cette fonction est utilisable uniquement pour les personnes dites « admin ». Cette fonction ne fonctionne pas en message privé. Elle ne permet pas de bannir les admins du serveur.

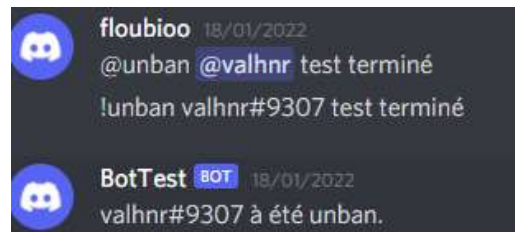
Exemple :



Commande : **!unban** [**toto#3124**] [**raison**]

Description : Cette commande permet de dé bannir une personne du serveur.

Contrainte(s) : Cette fonction est utilisable uniquement pour les personnes dites « admin ». Cette fonction ne fonctionne pas en message privé.



Commande : **!info**

Description : Cette commande permet d'avoir des informations sur le serveur.

Contrainte(s) : Cette fonction ne fonctionne pas en message privé. Cette fonction prend en compte les salons supprimés et les membres bannis.



7 Explication des différents fichiers nécessaires pour ce projet :

7.1 ProjetBotDiscord.py :

Ce fichier est le programme python permettant de faire fonctionner le bot récemment créé. Il utilise les bibliothèques récemment ajoutées via les commandes pip pour réaliser les actions vues précédemment.

Présentation du code :

Dans un premier temps on initialise le bot discord en utilisant l'extension de l'API discord « **commands** ». On précise ensuite dans cette même initialisation que les messages commençant par « ! » sont destinés au bot. On indique ensuite une description facultative, dans notre cas se sera « Bot SDTS » puis on ajoute « l'intents » permettant au bot de récupérer les membres des serveurs dans lequel le bot a été implémenté.

Slash sera une variable qui utilisera la fonction « **SlashCommand** », cette fonction permet d'utiliser les boutons graphiques sur discord, cela nous servira pour l'implémentation d'une calculatrice virtuelle dans discord.

Une suite de liste est ensuite créée, parmi ces listes on retrouve les listes suivantes (listeAlpha, listeNbr, listeCarac) qui nous permettront de générer des suites de caractères afin de générer des mots de passe. La liste « listeAdmin » contiendra le pseudo des différents utilisateurs admin.

```
18 import discord
19 from neuralintents import GenericAssistant
20 import json
21 import deepl
22 import random
23 import requests
24 from math import *
25 from discord.ext import commands
26 from discord_slash import ButtonStyle, SlashCommand
27 from discord_slash.utils.manage_components import *
28
29 #Création de la variable bot en indiquant que les commandes destinées au bot commenceront par "!"
30 # intents=discord.Intents.all() -> pour permettre au bot de récupérer les membres des serveurs
31 bot = commands.Bot(command_prefix = "!", description = "Bot SDTS", intents=discord.Intents.all())
32 #slash permet d'utiliser des boutons et des listes
33 slash = SlashCommand(bot, sync_commands=True)
34 #Création et initialisation des variables de type liste
35 listeAlpha = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
36 listeAdmin = []
37 listeNbr = ['1','2','3','4','5','6','7','8','9','0']
38 listeCarac = ['/','@','*','+','-','?',';','!']
```

La fonction **lancementBot** permet de renseigner les différentes informations essentielles pour l'exécution du programme c'est-à-dire que l'on demande dans un premier temps à l'utilisateur « admin » quel est le TOKEN du bot créé afin de relier notre programme au bot créé précédemment puis nous cherchons ensuite à savoir quels sont les utilisateurs admins. Ces utilisateurs pourront donc réaliser des tâches impactant d'autres individus sur un même serveur.

```
40 def lancementBot():
41     fin = False
42     idBot = ""
43     #id ou TOKEN est récupérable sur la page web developpeur de discord après avoir créé le bot
44     idBot = str(input("Veuillez renseigner l'id (TOKEN du bot) : "))
45
46     while fin == False:
47         retour = str(input("Qui peut avoir le droit admin (ex : TOT0#212)\nPour quitter utiliser le chiffre '0'\nRenseigner l'identifiant : "))
48         if retour != "0":
49             listeAdmin.append(retour)
50             print(listeAdmin)
51         else:
52             fin = True
53             try:
54                 bot.run(idBot)
55             except:
56                 print("idBot non correct")
```

Def IAchatBot est la fonction qui sera appelée lorsque le bot recevra un message commençant par « !chat ». Cette fonction permettra de générer un message présent dans intents.json par rapport au tag (message) renseigné par l'utilisateur.

La variable assistant est la variable qui utilisera neuralintents, on précise via la fonction GenericAssistant de la librairie que le fichier intents.json sera notre « base de données », c'est à dire que c'est avec ce fichier de données que nous allons entrainer notre bot et faire fonctionner notre « intelligence artificielle ».

La fonction train_model entrainera le programme sur les données du fichier intents.json et save_model sera utilisé pour sauvegarder cet entrainement.

Note : A chaque démarrage du bot, l'entrainement du bot se lancera.

```
58 def IAchatBot(message):
59     retour = ""
60     print(assistant.request(message))
61     retour = assistant.request(message)
62     return retour
63
64 assistant = GenericAssistant('intents.json', model_name="test_model")
65 assistant.train_model()
66 assistant.save_model()
```

L'API discord.py utilise le système de **coroutine** de python c'est-à-dire qu'on utilise **async** et **await** pour réaliser des tâches. Cela permet de planifier des séries de tâches. Les coroutines sont donc automatiquement planifiées pour s'exécuter à un moment donné. Le **await** permet au programme d'arrêter l'exécution d'une fonction, ce qui permet alors au bot de faire plusieurs actions en même temps sans utiliser la notion de multi processus (thread). J'utilise @bot.event pour déclarer les actions à effectuer lorsque les événements suivants appariassent :

On_ready signifie que le bot est prêt à réaliser différentes actions en fonction des différents événements possibles (commandes des utilisateurs). Le bot a donc préparé les données reçues de discord cela veut dire que la connexion aux serveurs a correctement fonctionné.

On_connect signifie que le bot est en ligne sur discord, il s'est donc connecté avec succès à discord.

On_disconnect signifie que le bot s'est déconnecté correctement de discord. Si la connexion a échoué sur discord alors l'évènement se déclenchera.

```
71 @bot.event
72 async def on_ready():
73     print("Bot prêt")
74 #S'active automatiquement lorsque le bot est connecté .event fait partie des différents événements de la librairie discord.py
75 @bot.event
76 async def on_connect():
77     print("Bot connecté")
78
79 #S'active automatiquement lorsque le bot est déconnecté
80 @bot.event
81 async def on_disconnect():
82     print("Bot déconnecté")
83
```

Dans la suite du code nous retrouverons beaucoup de « @bot.command() », cela permet d'utiliser l'extension de l'API discord nommée « Commands » afin de définir des fonctions (suite de tâches) reconnues par le bot en utilisant le système de coroutine.

Exemple d'utilisation de `@bot.command()` et des coroutines :

`@bot.command()`

Async def message(ctx) :

`Await ctx.send(« Hello World »)`

Dans cet exemple la fonction message est définie comme commande, c'est-à-dire qu'on peut demander au bot de réaliser la fonction message. Lorsque le bot recevra le message !message il exécutera donc la fonction du même nom que le message reçu par le bot. `Ctx.send()` permet au bot de répondre « Hello World » dans le Channel ou il a reçu « !message ». En effet le paramètre `ctx` représente le contexte de l'évènement c'est-à-dire que le bot peut savoir dans quel Channel le message « !message » a été envoyé. Nous pouvons donc dire que le système de coroutine couplé au système de `@bot.command()` permet d'effectuer de la programmation événementielle.

Pour la fonction **aide**, on affiche dans le Channel du message reçu les différentes actions pouvant être réalisées.

```
85 @bot.command()
86 async def aide(ctx):
87     com = ""
88     com = "linfo\nldire [expression]\nltraduireEN [expression]\nltraduireFR [expression]\nltraduireES [expression]\nltraduireDE [expression]"
89     #fonction send du contexte pour renvoyer un message dans le meme channel que le message envoyé par l'utilisateur
```

La fonction **genererP** est une fonction prenant en paramètre le nombre de caractères sur lequel nous souhaitons générer un mot de passe. Nous vérifions tout d'abord que le message reçu ne se trouve pas dans un serveur discord, le but étant de garder le mot de passe généré privé. Ensuite on génère un nombre aléatoire allant de 0 à 3. Si le nombre aléatoire généré est 1 alors nous générons un chiffre aléatoirement présent dans la liste « `listeNbr` ». Si le nombre choisi est 2 alors nous générons un caractère spécial présent dans la liste « `listeCarac` », si le chiffre aléatoire est 3 alors on génère un caractère alphabétique en majuscule sinon nous générons un caractère alphabétique en minuscule.

```
93 @bot.command()
94 async def genererP(ctx, nbrC):
95     # ctx.guild retourne une valeur uniquement si le message envoyé se trouve dans une guild 'serveur'
96     if not ctx.guild:
97         #si le paramètre nbrC est >= 6 on répartie la taille du mot de passe pour les majuscules, les minuscules, les caractères spéciaux et les chiffres
98         if int(nbrC) >= 6:
99             nbrDivise = nbrC // 5
100             nbrChiffre = ceil(int(nbrC)/5)
101             nbrCaractere = ceil(int(nbrC)/5)
102             nbrMajuscule = ceil(int(nbrC)/5)
103             nbrMin = int(nbrC) - (nbrChiffre + nbrMajuscule + nbrCaractere)
104             i = 0
105             pos = 0
106             password = [0] * int(nbrC)
107             retour = ""
108             while (i < int(nbrC)):
109                 # permet de générer aléatoirement la position des différents caractères
110                 r = random.randint(0, 3)
111                 if (r == 1 and nbrChiffre >= 1):
112                     # prend une nouvelle valeur, la valeur aléatoire est ensuite la position du caractère à utiliser dans la liste nommée listeNbr
113                     r = random.randint(0, len(listeNbr)-1)
114                     password[pos] = listeNbr[r]
115                     nbrChiffre = nbrChiffre - 1
116                     pos += 1
117                     i += 1
118                 elif (r == 2 and nbrCaractere >= 1):
119                     r = random.randint(0, len(listeCarac)-1)
120                     password[pos] = listeCarac[r]
121                     nbrCaractere = nbrCaractere - 1
122                     pos += 1
123                     i += 1
124                 elif (r == 3 and nbrMajuscule >= 1):
125                     r = random.randint(0, len(listeAlpha)-1)
126                     password[pos] = listeAlpha[r].upper()
127                     nbrMajuscule = nbrMajuscule - 1
128                     pos += 1
129                     i += 1
130                 else:
131                     if (nbrMin >= 1):
132                         r = random.randint(0, len(listeAlpha)-1)
133                         password[pos] = listeAlpha[r]
134                         nbrMin = nbrMin - 1
135                         pos += 1
136                         i += 1
137             # join permet de réunir les différents éléments de la liste
138             retour = "Le mot de passe généré est : || "+ " ".join(password)+ "||"
139             print(password)
140             await ctx.send(retour)
141         else:
142             retour = "Les mots de passe se génèrent sur 6 caractères minimums"
143             await ctx.send(retour)
144         else:
145             await ctx.send("La fonction 'lgenererP' fonctionne uniquement en message privé")
```

Etudiant : Vaulry Yannis

Master : SDTS FA

Rapport créé le : 14/02/2022

La fonction **calculatrice** permet de générer virtuellement une calculatrice pour réaliser des calculs « simples ». Nous vérifions tout d'abord que la commande reçue est bien une commande envoyée en message privé. Si ce message est un message privé nous créons 4 listes : buttons, buttons2, buttons3, buttons4. Dans ces listes nous créons des boutons grâce à la librairie **discord_slash**. On précise pour ces boutons le style soit la couleur des boutons, le label soit la chaîne de caractères qui sera présente sur le bouton et le custom_id qui nous permettra de différencier les différents boutons d'une même liste.

Nous créons ensuite 4 actionrow qui correspondront aux lignes des différentes actions possibles. Cela permet de faire un retour à la ligne pour chaque ligne actionrow. Nous avons donc 4 lignes d'actions comprenant au minimum 4 boutons.

Ensuite, nous affichons le message « Calculatrice : » puis nous plaçons dans discord nos différents composants. Une fois cela fait la calculatrice virtuelle est disponible et visible sur discord.

Tant que l'utilisateur n'appuie pas sur le bouton « fin » de la calculatrice alors le bot continuera d'attendre des actions de sa part. Pour le premier bouton utilisé par l'utilisateur nous vérifions que ce dernier correspond bien à un chiffre. Si ce n'est pas le cas on retourne un message d'alerte. Donc si le premier bouton utilisé par l'utilisateur correspond à un chiffre alors on laisse la possibilité à l'utilisateur de renseigner d'autres chiffres pour former un nombre. Lorsque l'utilisateur utilisera un bouton avec un label de calcul (+, -, *, /) alors on attendra que l'utilisateur renseigne à nouveau un chiffre ou un nombre avant de retourner le résultat du calcul dans le message original « Calculatrice : » de la fonction calculatrice. Lorsque le résultat est affiché sur discord, l'utilisateur peut alors utiliser le bouton « fin » pour terminer le programme ou renseigner un nouveau nombre ou chiffre pour effectuer un nouveau calcul.

Voir le fichier ProjetBotDiscord (fonction trop longue).

La fonction **dire** permet de faire répéter la liste « texte » par le bot, c'est à dire que le robot renvoie le message envoyé par un utilisateur (le message envoyé par l'utilisateur est récupéré dans la variable « texte »). Cette fonction agit dans notre code comme une fonction de test.

```
444 @bot.command()
445 async def dire(ctx, *texte):
446
447     await ctx.send(" ".join(texte))
```

Les fonctions **traduireES**, **traduireFR**, **traduireEN** et **traduireDE** sont des fonctions permettant d'utiliser l'API DEEPL. Ces fonctions utilisent la liste « message » que les utilisateurs renseignent pour ensuite traduire dans la langue souhaitée cette liste via la méthode translate_text de l'API.

Attention, dans le code du fichier « ProjetBotDiscord.py », la valeur donnée en paramètre des fonctions Translator est un TOKEN, si vous avez créé un compte DEEPL il est alors possible d'utiliser le vôtre sinon le TOKEN du code par défaut suffira pour réaliser des traductions.

```
461 @bot.command()
462 async def traduireES(ctx, *message):
463     message = " ".join(message)
464     message = message.replace("'", ' ')
465     translator = deepl.Translator("3574d2a4-ce63-a220-edb5-a336302ba645:fx")
466     traduction = translator.translate_text(message, target_Lang="ES") # -> ESPAGNOL
467     print(traduction)
468     await ctx.send(traduction)
```


La fonction **nettoyer** permet aux utilisateurs « admin » de nettoyer une partie des messages textuels d'un Channel. Pour utiliser cette fonction les administrateurs doivent renseigner en paramètre le nombre de message à supprimer. Le bot se chargera par la suite de supprimer les messages jusqu'à ce que la condition du nombre de message à supprimer soit respectée. Cette fonction utilise donc la « listeAdmin » dans laquelle vous avez lors du lancement du bot renseigné les pseudos des différents utilisateurs de types « admin ». Cette fonction utilise donc le contexte actuel (Channel actuel) pour ensuite utiliser la fonction history et la fonction flatten pour récupérer l'historique des derniers messages du Channel en question.

```

489 @bot.command()
490 async def nettoyer(ctx, nombre : int):
491     if ctx.guild:
492         #Si l'utilisateur fait partie de la liste Admin générée lors de l'appel de la fonction lancementBot() alors on supprime les lignes demandées par ce dernier
493         if str(ctx.message.author) in listeAdmin:
494             messages = await ctx.channel.history(limit = nombre).flatten()
495             for messages in messages:
496                 await messages.delete()
497         else:
498             await ctx.send("'!nettoyer' est utilisable uniquement par les profils Admin")
499     else:
500         messages = await ctx.channel.history(limit = nombre).flatten()
501         for messages in messages:
502

```

La fonction **stationsT** est une fonction qui servira notamment pour récupérer les différents noms des stations de métros, de bus et de tramways selon une ligne donnée. La fonction prend en paramètre le type de transport (métros, tramways, bus) et le numéro de transport (exemple : 4 pour le métro 4). Si nous analysons le code, nous pouvons voir que cette fonction utilise l'API RATP, cette API est une API en ligne, elle n'est pas directement utilisable dans python sous la forme de librairie. Il est donc nécessaire de lancer une requête HTTP pour récupérer le retour de l'API. Le retour de l'API est un retour au format JSON donc pour pouvoir mieux récupérer les données souhaitées nous générons une variable « contenu » qui sera au format JSON. Avec ce format nous pourrions récupérer les stations des différents transports avec cette ligne de code :

`contenu["result"]["stations"][i]["name"]`

```

505 @bot.command()
506 async def stationsT(ctx, typeTrans, numTrans):
507     r = ""
508     nom = ""
509     requete = "https://api-ratp.pierre-grimaud.fr/v4/stations/" + typeTrans + "/" + numTrans + "?way=A"
510     #Récupération des valeurs de retours de la requete
511     retour = requests.get(requete)
512     #Conversion de la variable de type String en json
513     contenu = retour.json()
514     #Récupération des différents éléments de contenu (json)
515     for i in range(1,25):
516         nom = contenu["result"]["stations"][i]["name"]
517         r = r + nom + "\n"
518     if (retour != ""):
519         await ctx.send(r)
520     else:
521         retour = "Les paramètres ne sont pas corrects"
522         await ctx.send(r)

```

La fonction **transport** est une fonction qui comme la fonction **stationsT** utilise l'API RATP. Cette dernière prend en paramètre le type de transport, le numéro de transport ainsi que la position de départ souhaitée soit la station de départ (Pour savoir quel est le nom d'une station, vous pourrez donc utiliser la méthode présentée précédemment soit la méthode **stationsT**). Cette fonction agit comme **stationsT**, elle utilise une variable de type JSON.

```

526 @bot.command()
527 async def transport(ctx, typeTrans, numTrans, *depart):
528     #%20 permet de remplacer les différents espaces de la variable depart pour y mettre %20 (cela est nécessaire pour la requete HTTPS)
529     depart = "%20".join(depart)
530     #On prend en compte les ' et / pour éviter des problèmes de transmission lors de la requete HTTPS
531     depart = depart.replace("'", '\')
532     depart = depart.replace("/", '/')
533     #Requete envoyée à l'API
534     requete = "https://api-ratp.pierre-grimaud.fr/v4/schedules/" + typeTrans + "/" + numTrans + "/" + depart + "/A%2BR"
535     #Récupération des valeurs de retours de la requete
536     retour = requests.get(requete)
537     #Conversion de la variable de type String en json
538     contenu = retour.json()
539     #Récupération des différents éléments de contenu (json)
540     if contenu["result"][0]["message"] != "Schedules unavailable":
541         directionA = contenu["result"][0]["destination"]
542         directionB = contenu["result"][2]["destination"]
543         tempsA = contenu["result"][0]["message"]
544         tempsB = contenu["result"][2]["message"]
545         #r correspond à la valeur de retour renvoyé à l'utilisateur
546         r = "Prochain pour aller à : " + directionA + " est dans " + tempsA + "\nProchain pour aller à : " + directionB + " est dans " + tempsB + ""
547         await ctx.send(r)
548     else:
549         await ctx.send("Les paramètres ne sont pas corrects")

```

La fonction **chat** est la fonction qui utilise neuralintents, cette fonction appelle la fonction générée au début du code « **IAchatBot** » pour générer un message en fonction du tag présent dans « le pattern » renseigné par l'utilisateur.

```

551 @bot.command()
552 async def chat(ctx, *message):
553     message = " ".join(message)
554     retour = ""
555     retour = IAchatBot(message)
556     await ctx.send(retour)
557

```

Les fonctions **creerCT** et **creerCV** permettent de créer des channels textuels et vocaux. Ces fonctions prennent en paramètre le nom du Channel à créer. Nous utilisons les fonctions `create_voice_channel` et `create_text_channel` sur le contexte du serveur Discord.

Cette fonction échoue si le message reçu par le bot est un message privé ou si le message reçu provient d'un utilisateur non admin.

```

558 #CreerCT permet de créer des salons de type texte si l'utilisateur se trouve dans un serveur et si l'utilisateur est un admin
559 @bot.command()
560 async def creerCT(ctx, *nom, reason = None):
561     if ctx.guild and str(ctx.message.author) in listeAdmin:
562         nom = " ".join(nom)
563         channel = await ctx.guild.create_text_channel(nom)
564     else:
565         await ctx.send("'!creerCT' est utilisable uniquement lorsque le bot se trouve dans un serveur et est utilisable uniquement par les profils Admin")
566 #CreerCV permet de créer des salons vocaux si l'utilisateur se trouve dans un serveur et si l'utilisateur est un admin
567 @bot.command()
568 async def creerCV(ctx, *nom, reason = None):
569     print(ctx.message.author)
570     if ctx.guild and str(ctx.message.author) in listeAdmin:
571         nom = " ".join(nom)
572         channel = await ctx.guild.create_voice_channel(nom)
573     else:
574         await ctx.send("'!creerCV' est utilisable uniquement lorsque le bot se trouve dans un serveur et est utilisable uniquement par les profils Admin")

```


La fonction **suppC** permet de supprimer un Channel d'un serveur. Elle prend en paramètre le nom du Channel à supprimer et elle nécessite pour être utilisée que le message reçu par le bot provienne d'un serveur discord et que son expéditeur soit un membre de la « listeAdmin ».

```
576 @bot.command()
577 async def suppC(ctx, *nom, reason = None):
578     if ctx.guild and str(ctx.message.author) in listeAdmin:
579         nom = " ".join(nom)
580         channel = discord.utils.get(ctx.guild.channels, name=nom)
581         if channel is not None:
582             await channel.delete()
583         else:
584             await ctx.send("Le channel n'existe pas")
585     else:
586         await ctx.send("'!suppC' est utilisable uniquement lorsque le bot se trouve dans un serveur et est utilisable uniquement par les profils Admin")
```

La fonction **info** permet d'afficher des informations sur le serveur pour cela nous utilisons la variable gérée au début du code soit la variable « bot » qui contient les informations sur les serveurs dans lesquels le bot est présent. Cette fonction agit sur le serveur dans lequel le message : !info a été envoyé, on récupère ensuite le nombre de Channels vocaux via `voice_channels` et le nombre de channels textuels via `text_channels`.

```
588 @bot.command()
589 async def info(ctx):
590     if ctx.guild:
591         channelT = 0
592         channelV = 0
593         membre = 0
594         for guild in bot.guilds:
595             for channel in guild.voice_channels:
596                 channelV += 1
597             for channel in guild.text_channels:
598                 channelT += 1
599             for member in guild.members:
600                 membre += 1
601             membre = membre - 1
602         if channelV > 1:
603             retour = "Il y'a " + str(channelV) + " salons vocaux\n"
604         else:
605             retour = "Il y'a " + str(channelV) + " salon vocal\n"
606
607         if channelT > 1:
608             retour = retour + "Il y'a " + str(channelT) + " salons de type texte\n"
609         else:
610             retour = retour + "Il y'a " + str(channelT) + " salon de type texte\n"
611
612         if membre > 1:
613             retour = retour + "Il y'a " + str(membre) + " membres dans le serveur : " + guild.name
614         else:
615             retour = retour + "Il y'a " + str(membre) + " membre dans le serveur : " + guild.name
616
617         await ctx.send(retour)
618     else:
619         await ctx.send("'!info' est utilisable uniquement lorsque le bot se trouve dans un serveur")
```

Les fonctions **ban** et **unban** prennent en paramètre le pseudo d'un utilisateur (@pseudo) ainsi qu'une raison soit un justificatif de ban ou de déban.

```

623 @bot.command()
624 async def ban(ctx, user : discord.User, *reason):
625     if ctx.guild and str(ctx.message.author) in listeAdmin:
626         reason = " ".join(reason)
627         await ctx.guild.ban(user, reason = reason)
628         await ctx.send(f"{user} à été ban pour la raison suivante : {reason}.")
629     else:
630         await ctx.send("Vous n'avez pas les droits")
631
632 #Unban permet de débannir un utilisateur en donnant une raison
633 @bot.command()
634 async def unban(ctx, user, *reason):
635     if ctx.guild and str(ctx.message.author) in listeAdmin:
636         reason = " ".join(reason)
637         userName, userId = user.split("#")
638         bannedUsers = await ctx.guild.bans()
639         for i in bannedUsers:
640             if i.user.name == userName and i.user.discriminator == userId:
641                 await ctx.guild.unban(i.user, reason = reason)
642                 await ctx.send(f"{user} à été unban.")
643                 return
644             #Ici on sait que l'utilisateur n'a pas été trouvé
645             await ctx.send(f"L'utilisateur {user} n'est pas dans la liste des bans")
646     else:
647         await ctx.send("Vous n'avez pas les droits")
648

```

Le fonctionnement du bot se fait via la fonction lancementBot vu précédemment :

```

650 #Appel de la fonction pour lancer le bot
651 lancementBot()

```

7.2 intents.json :

Le fichier intents.json est un fichier « modèle » que nous allons utiliser pour entraîner notre bot sur différentes patterns et tags. Ce fichier comporte une structure de type JSON et est utilisé pour la librairie neuralintents. Voici ci-dessous un aperçu du fichier.

```

{"intents": [
  {
    "tag": "salutation",
    "patterns": ["Bonjour", "Salut", "Hey", "Hello", "Salutation", "Whats up", "Yo", "Hola"],
    "responses": ["Hello !", "Salut !", "Bonjour :)"],
    "context_set": ""
  },
  {
    "tag": "adieu",
    "patterns": ["Adieu", "Au revoir", "Goodbye", "Bonne journée", "Bonne soirée", "bye", "cao", "a plus"],
    "responses": ["On se reparlera plus tard :)", "Au revoir", "bonne journée", "a demain", "a plus", "cao", "Bonne soirée", "Goodbye", "a plus tard", "a tout a l'heure"],
    "context_set": ""
  }
],

```

Il est donc possible à tout moment d'ajouter de nouveaux tags pour entraîner le bot récemment créé.

8 Conclusion :

J'ai apprécié réaliser ce projet en python, discord est une plateforme que j'utilise régulièrement mais je ne me suis jamais penché sur le sujet « création d'un bot discord ». J'ai découvert en utilisant l'API discord et ses extensions des actions que je ne pensais pas possible notamment pour la génération de bouton avec discord_slash. Actuellement, j'utilise même ce projet pour un de mes serveurs discord. J'ai aussi apprécié voir le bot répondre à des phrases dites simples grâce à la fonctionnalité chat utilisant neuralintents. J'aurais aimé ajouter dans le fichier intents.json d'autres patterns et d'autres Tag pour que le bot puisse réellement tenir tous types de conversations, je pense que c'est une amélioration qui aurait été nécessaire. Ce projet m'a aussi mis en difficulté sur la gestion des erreurs notamment pour la fonction calculatrice et pour la gestion des caractères spéciaux envoyés au bot. Je n'ai pas trouvé de moyen pour interdire le caractère ' »' car en effet, lorsqu'une fonctionnalité est utilisée et que ce caractère est en paramètre nous pouvons voir dans le cmd ou terminal que des exceptions surviennent sans qu'un message de retour soit envoyé à l'utilisateur. Utilisant des API (DEEPL et RATP), le programme n'est pas auto dépendant ce qui n'est pas une chose que je souhaitais mais malgré ces imprévus j'ai appris beaucoup de choses sur l'application discord en allant regarder dans la documentation de l'API et j'ai aussi appris des fonctionnalités de python que je ne connaissais pas auparavant.