Rapport Python outil de l'ingénieur

TP4 : Bot Discord

Julien Voeung U32

Objectif

Ce TP avait pour but de créer un bot discord en python et ayant comme spécificités :

- Lancement en CLI du bot avec passage de paramètres via la console
- Un fichier de configuration pour le bot
- Journalisation des actions du bot dans un fichier de logs
- Une liste de commandes supportées par le bot avec un !help qui liste les commandes
- Chercher et afficher le résultat de géolocalisation d'une IP sur Shodan, Calculer la suite de Syracuse d'un entier et afficher le résultat sur le salon discord, calculer la multiplication égyptienne de deux entiers
- Tout en programmation Orientée Objet

Explication du code

Les fichiers sont au nombre de 5 avec 4 fichiers .py, qui seront les 4 classes utilisés et 1 fichier .json qui sera le fichier de configuration du bot. L'articulation du projet se fait de la manière suivante, la classe MyBot (qui se trouve dans le fichier bot.py) contrôle et fait l'appel des fonctionnalités. Chaque fonctionnalité du bot est associée à une classe, ici nous avons trois fonctionnalités qui sont la recherche d'une adresse ip, la suite de Syracuse et la multiplication égyptienne. Une fonctionnalité est représentée par une classe. Ainsi tous nos fichier python sont des classes, le main se trouve dans le fichier bot.py, qui est le tronc de notre projet.

Le fichier json est un fichier de configuration pour l'utilisateur qui lance le bot, il contient les informations que l'on souhaite utilisés et qui ne seront pas codé en dur dans le code python par soucis de sécurité. On retrouvera la clé Api Shodan et le token du bot discord.

```
{
    "apiKey": "ttIZacJo0FkHUDDzuLimF8ku48YAXHzP",
    "botToken": "OTg4OTE1MjQ0NDA5Njg4MDc0.Gr2_A7.jaoZTCDuIEGJ3ieq1QJq2P8OXwHTwMF3zpFfms"
}
```

Dans le fichier principal, c'est-à-dire le fichier bot.py, il contient une classe MyBot et un main. C'est donc ce fichier par lequel on lancera notre bot. L'exécution se fait ainsi, on a un argument -c ou -config puis notre paramètre qui est le fichier de configuration json.

```
(my_env) PS C:\Users\voeun\python_inge\TP4> python bot.py -c configfiles.json
```

Tout commence à ce niveau, avant le main et avant la classe bot. On commence par crée un client, qui sera notre bot car il agira comme un client sur discord. Puis on a la méthode parse_args, qui sera appelé juste après. Donc on parse la commande qui a été passé dans la console, pour récupérer le nom du fichier de configuration. De là, on charge les données JSON qui sont contenue dans le fichier, dans une variable data.

En la déclarant ainsi, on accède facilement au valeur JSON sans soucis d'incompatibilité dans le code.

```
#fichier log
logging.basicConfig(filename='tracebot.log', encoding='utf-8', level=logging.DEBUG)

#creation d'un client discord
client = discord.Client()

#parse pour recuper le nom du fichier de configuration JSON en parametre
def parse_args() -> Namespace:
    parser = ArgumentParser()
    parser.add_argument(
        "-c", "--config", help="Config file", required=True, dest="config"
    )
    return parser.parse_args()

#parse
args = parse_args()
#chargement dans data des donnes JSON
with open(args.config) as json_data_file:
    data = json.load(json_data_file)
logging.info('Parse sur le fichier de configuration JSON')
```

L'étape d'après est le main, on déclare un objet MyBot dans une variable bot. Puis on le lance via une méthode de la classe, qui elle-même exécute la méthode run sur le client qui est le bot.

```
#main
if __name__ == "__main__":
    #creation d'un objet bot
    bot = MyBot()
    #lancement du bot avec methode launch
    bot.launch()
```

```
#pour demarer le bot
def launch(self):
    client.run(data['botToken'])
    logging.info('Lancement du BOT')
```

L'étape suivante est les méthodes async, qui sont là pour gérer les événements du client comme la connexion avec on_ready et l'écoute des messages sur un channel avec on_message.

```
#methode event du bot discord connexion
@client.event
async def on_ready():
    channel = client.get_channel(986278767473336373)
    await channel.send("BoBOT is connected ! Tape **!help** for more informations")
    logging.info('BOT is connected')
```

Pour les écouter les commandes passés dans le channel, on utilise la méthode startwith, donc tous les message commençant par la commande lié à la fonctionnalités exécuteront un processus associé.

```
#methode event du bot discord message
@client.event
async def on_message(message):
    #commande !iplocation
    if message.content.startswith('!iplocation'):
        logging.info('Traitement de la commande !iplocation')
        #split pour separer les parametres passes
        response = message.content.split(" ")
```

Ensuite, on déclare l'objet en fonction de la fonctionnalité qu'on souhaite utilisé, ici c'est la commande sur shodan. On crée donc un objet func_shodan, qu'on instancie avec les paramètres qui ont été récupéré précédemment. Puis on appelle la méthode qui exécute la fonctionnalité, ici on rentre en paramètre l'api key qui se trouve dans notre variable data et qui se trouve dans le même fichier.

Finalement, on envoie dans le channel un message avec le résultat obtenue. Via la méthode send.

```
else:
    logging.info('Traitement de la requete shodan')
    #creation d'un objet func_shodan
    shodan = func_shodan(response[1],response[2])
    #appel de la methode pour requeter shodan
    shodan.serveur_shodan(data['apiKey'])
    #message sur le channel de l'url
    await message.channel.send(shodan.urlMap)
    logging.info('Traitement de la commande !iplocation terminé')
```

La méthode serveur_shodan est appelé pour exécuter une requête sur l'api shodan.

```
#methode pour la requete shodan
def serveur_shodan(self,apiKey):
    #utilisation de la methode host pour effectuer une r

    api = shodan.Shodan(apiKey)
    logging.info('connexion a api shodan via apikey')
```

Voici un exemple de lancement avec la commande !help et la commande !iplocation.