FICHE TECHNIQUE

Nous avons choisi de réaliser ce chatbot dans le langage Python car ce dernier est répandu, populaire, et dispose d'une documentation très conséquente.

De plus, il est important de créer le bot sur un noyau Windows et non Linux afin de ne pas générer de conflits pendant le déploiement (en effet, il semble que le cloud de Microsoft n'accepte que des chatbots ayant été créés sur un noyau Windows). Il faudra donc utiliser le cmd de Windows pour lancer les différentes commandes.

CRÉATION ET TEST EN LOCAL

Dans un premier temps, ouvrez votre invite de commande Windows, et vérifiez que Python est bien installé. Attention, les dernières versions ne fonctionneront pas avec ce produit, il faut donc que vous ayez la version 3.6 installée. Lors de l'installation, cochez la case "Ajouter au path". Vous pouvez poursuivre dès que l'installation est achevée.

Plusieurs packages Python vont être nécessaires pour créer le bot. Voici donc la liste des installations à effectuer :

```
pip install botbuilder-core
pip install asyncio
pip install aiohttp
pip install cookiecutter==1.7.0
```

Il est conseillé d'écrire tous les packages requis dans le fichier requirements.txt. Cela sera indispensable lors du déploiement car le bot ne sera plus sur votre machine où tous les packages sont déjà installés, il devra donc savoir de quels packages il a besoin.

Si vous effectuez ces commandes via le cmd, il faut trouver un moyen de lancer des commandes Python. Pour ma part, j'utilise le préfixe `python - m` au début de chaque commande.

Enfin, il ne vous reste plus qu'à récupérer un template de bot (EchoBot dans le cas présent) et de le télécharger en local grâce à la commande suivante :

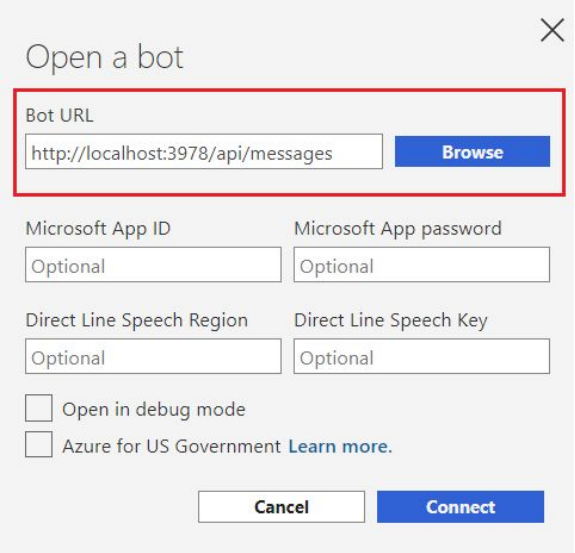
```
python -m cookiecutter
https://github.com/microsoft/botbuilder-python/releases/download/Templat
es/echo.zip
```

Vous n'aurez plus qu'à renseigner le nom souhaité pour le chatbot et le tour est joué. Il ne reste plus qu'à tester le bot en local pour vérifier que tout fonctionne correctement. Pour cela, vous aurez besoin d'installer le Bot Framework Emulator (C'est ici que ça se passe : https://github.com/Microsoft/BotFramework-Emulator/blob/master/README.md).

Une fois téléchargé, vous n'avez plus qu'à démarrer le chatbot en lançant le app.py :

`python app.py` (ou simplement en lançant via l'IDE si vous en utilisez un.)

Le terminal vous affiche alors sur quel port a été lancé votre bot. Il ne reste plus à présent qu'à se rendre sur le Bot Framework Emulator et renseigner les informations nécessaires pour voir la magie opérer.



DÉPLOIEMENT

1) Premier déploiement

Dans cette partie, nous allons utiliser le cloud de Microsoft : Microsoft Azure. Afin de communiquer avec lui en ligne de commandes, il va être nécessaire d'installer Azure CLI (Command Line Interface). Pour cela, direction le lien suivant :

https://docs.microsoft.com/fr-fr/cli/azure/install-azure-cli-windows?tabs=azure-cli

Un fois téléchargé, vous pourrez effectuer toutes les commandes Azure dans le cmd (il faut bien entendu avoir un compte Azure pour cela). Ensuite plus qu'à se connecter à son compte Azure grâce à la commande suivante :

```
az login
```

Attention : il faut que sa session Azure soit active pour que la manœuvre fonctionne, il faut donc au préalable se connecter au portail Azure.

Cette commande vous emmènera sur l'interface web de Microsoft où il faudra vous connecter à votre compte. Une fois de retour sur le terminale de commande, vous pourrez accéder à la liste des abonnements (subscription) dont vous disposez sur votre compte avec la commande suivante :

```
az account list
```

Attention : notez bien l'id renvoyé par cette commande, celui-ci sera important pour la prochaine étape. Dans la plupart des cas, votre compte ne disposera que d'un seul abonnement.

Il faut à présent définir l'abonnement que nous souhaitons utiliser par défaut. Cela se fait grâce à la commande suivante (<azure-subscription> étant l'id obtenu lors de l'étape précédente) :

```
az account set --subscription "<azure-subscription>"
```

Vous devez à présent créer une "app registration" (enregistrer une application) sur Azure. C'est à partir de maintenant que les commandes deviennent légèrement plus

complexes. Voici comment procéder pour créer une “registration” :

```
az ad app create --display-name "<display-name>" --password "<password>"
--available-to-other-tenants
```

Ici, il faut choisir ce qu'on veut pour display name et password, le displayName étant le nom de l'application.

Cette commande renvoie un Json : il faut bien noter le appId qui s'y trouve car il sera utile par la suite (de même que le password que l'on vient de renseigner)

Courage, on touche presque au but !

Azure fonctionne avec des groupes de ressources qui permettent d’organiser ses entités dans le cloud. Il va donc à présent s’agir de placer cette application nouvellement créée dans un groupe de ressources. Pour cela, il faut donc créer au préalable ce groupe de ressources sur Azure, puis lancer la commande suivante :

```
az deployment group create --resource-group "<name-of-resource-group>"
--template-file "<path-to-template-with-preexisting-rg.json>"
--parameters appId="<app-id-from-previous-step>"
appSecret="<password-from-previous-step>" botId="<id or
bot-app-service-name>" newWebAppName="<bot-app-service-name>"
newAppServicePlanName="<name-of-app-service-plan>"
appServicePlanLocation="<region-location-name>" --name
"<bot-app-service-name>"
```

Pas de panique, voici à quoi correspondent les différents arguments présents dans la commande :

- `<name-of-resource-group>` : le nom du groupe de ressources créé au préalable
- `<path-to-template-with-preexisting-rg.json>` : il s’agit du chemin vers le fichier template-with-preexisting-rg.json qui se situe dans le dossier deploymentTemplates de votre bot.
- `<app-id-from-previous-step>` : tout simplement le appId obtenu lors des précédentes étapes
- `<region-location-name>` : dans notre cas, il faut choisir "westeurope"
- On peut choisir le même nom pour tous les autres paramètres, cela ne pose pas de problèmes. Personnellement, j’ai choisi le même nom que le <display-name> de l’app registration.

Enfin, dernière étape. Il faut maintenant compresser son code pour l'envoyer sur le cloud.

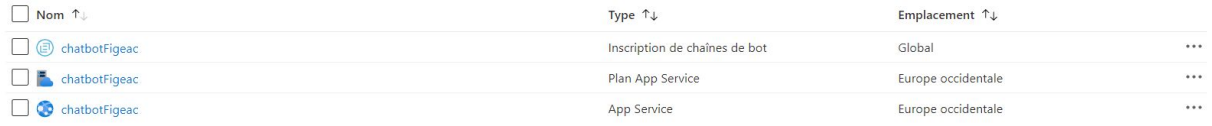
Attention : lors de la compression, faites bien attention à où se situent vos programmes python. En effet, il faut que ceux-ci soient à la racine de l'archive et non un cran plus bas.

Maintenant, plus qu'à lancer la commande suivant pour que le chatbot se déploie en quelques minutes :

```
az webapp deployment source config-zip --resource-group
"<resource-group-name>" --name "<name-of-web-app>" --src
"<project-zip-path>"
```

- `<name-of-resource-group>` : toujours le même nom du groupe de ressource créé précédemment
- `<name-of-web-app>` : le nom de la web app créée précédemment, à savoir le <bot-app-service-name> de la commande précédente
- `<project-zip-file>` : enfin, il faut mettre ici le chemin vers l'archive contenant votre code compressé.

L'opération prend alors quelques temps mais devrait se terminer sans encombre. Une fois fini, vous pouvez tester votre bot directement sur Azure. Pour cela, vous devrez vous rendre dans le groupe de ressources. Si tout s'est bien passé, 3 éléments ont été créés :



C'est dans le premier que vous devrez vous rendre pour tester votre bot à l'aide de la fonctionnalité "Tester dans une discussion web" qui s'y trouve.

Et voilà, votre chatbot est maintenant déployé et prêt à être utilisé sur différentes applications !

2) Redéploiement

Pour redéployer un chatbot sur Azure, rien de plus simple. Vérifiez tout d'abord que vous êtes bien connectés à Azure (en ligne puis avec la commande `az login`). Enfin il ne reste plus qu'à compresser à nouveau le code et à l'injecter sur le cloud avec la dernière commande :

```
az webapp deployment source config-zip --resource-group
"<resource-group-name>" --name "<name-of-web-app>" --src
"<project-zip-path>"
```

Il faut parfois attendre quelques minutes (1 ou 2 pour moi mais parfois plus) pour que le redéploiement soit effectif, puis vous pourrez tester le chatbot dans le chat web sur Azure pour vérifier que tout fonctionne correctement.

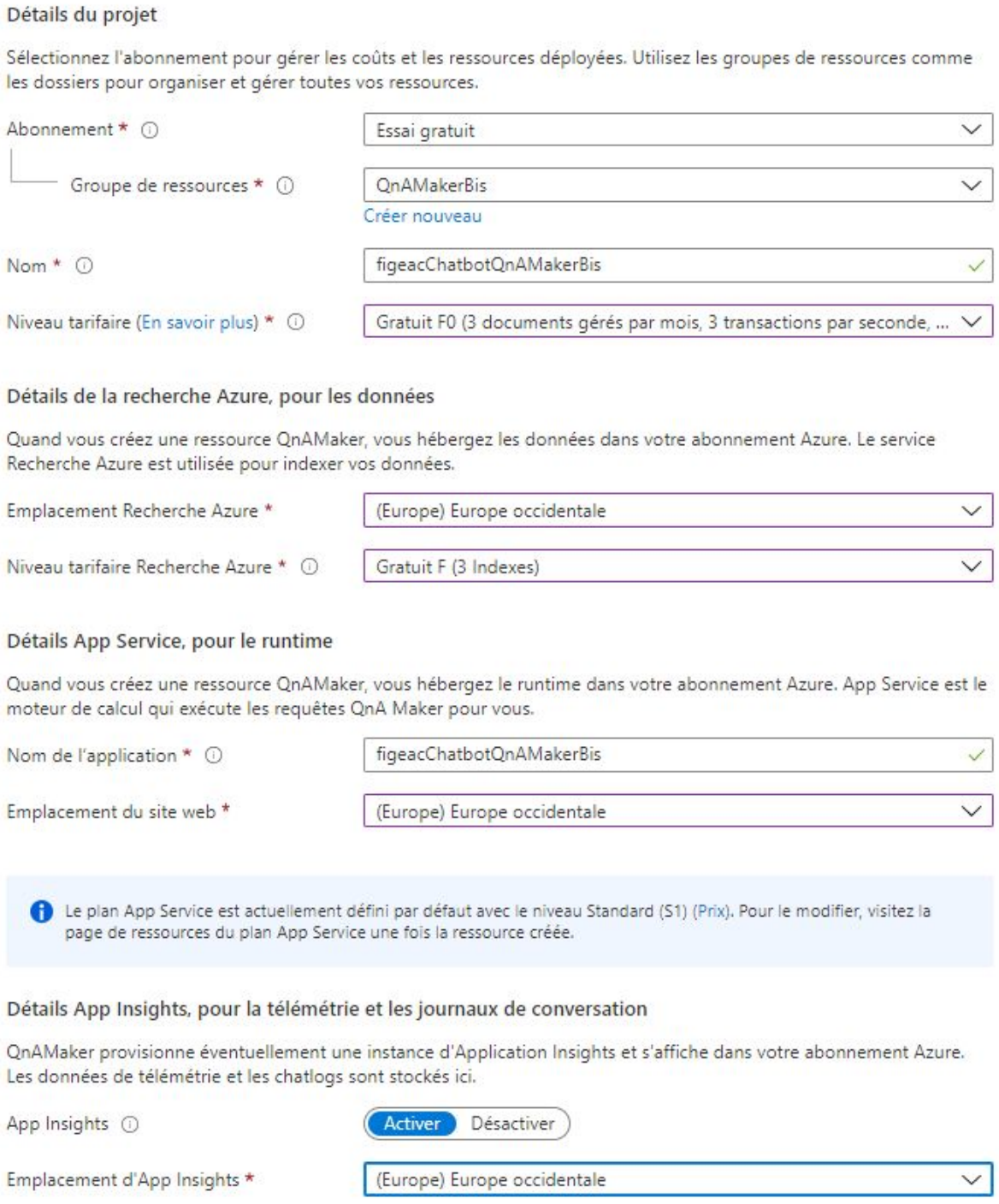
Si une erreur survient avec pour description "ImportError : no module named decorator", il vous faudra désinstaller puis réinstaller le client azure.

CRÉATION D'UN SERVICE DE QUESTION AND ANSWER (QNA MAKER)

La création d'un service QnA se fait en plusieurs temps. Il va d'abord falloir créer un "QnA maker service" associé à un groupe de ressources. Ce dernier permettra de créer une ou plusieurs bases de connaissances qui pourront être reliées au bot et dans lesquelles seront renseignées tous les couples de questions/réponses.

1) Création du QnA Maker service

La première étape sera de créer un QnA Maker service en se rendant sur ce lien : https://portal.azure.com/#create/Microsoft.CognitiveServicesQnAMaker. Une fois sur cette page, une multitude d'informations vous seront demandées. Tout d'abord, plusieurs options s'offrent à vous : vous pouvez choisir de créer un QnA Maker service managé ou non. La seule différence réside dans le fait que Azure gérera automatiquement certains paramètres ou non. Personnellement, j'ai expérimenté l'option non managée et elle fonctionne très bien.



Voici donc la liste de toutes les options que vous devrez remplir. Ici, j'ai choisi de placer le QnA Maker service dans un autre groupe de ressources, mais cela n'a pas vraiment d'importance pour la suite. Vous devrez donc choisir un nom pour votre QnA Maker puis renseigner les différentes options de votre abonnement (niveau tarifaire, emplacement géographique, ...). Pour nous, tout est au niveau tarifaire le plus bas et en Europe occidentale.

Enfin, Azure vous propose le choix d'activer ou non les app insights. Nous ne savons pas encore trop à quoi cela correspond mais nous avons gardé cette option activée dans le doute.

2) Création d'une base de connaissances

Une fois le QnA Maker créé, Azure vous renverra vers l'interface d'édition des knowledge bases. C'est ici que vous pourrez créer vos bases de connaissances et les remplir de questions/réponses. C'est donc ce que nous allons faire à présent : créez une base de connaissance et rentrer autant de questions/réponses que vous le souhaitez (une ou deux suffiront pour l'instant). Vous remarquerez que vous pouvez également renseigner des formulations alternatives à vos questions afin que le chatbot puisse mieux les comprendre.

Une fois cette étape effectuée, notre attention va se porter sur les deux boutons situés en haut à droite : "Save and train" et "Test". Tout d'abord, il vous faudra sauvegarder votre base de connaissance dans le cloud. Vous pourrez alors la tester pour vérifier que tout fonctionne correctement.

Une fois que vous êtes satisfait de votre base de connaissances, vous pourrez alors la publier (en cliquant sur l'onglet "Publish"). Après avoir confirmé la publication, vous verrez s'afficher un cadre contenant de précieuses informations : notez les bien ! Elles seront très utiles pour la suite.

Ces informations sont de la forme :

```
POST /knowledgebases/<knowledge-base-id>/generateAnswer
Host: <your-hostname>  // NOTE - this is a URL ending in /qnamaker.
Authorization: EndpointKey <qna-maker-endpoint-key>
```

Par souci de sûreté, nous vous invitons à vérifier que la souscription à ce service est bien passé en gratuit. Pour cela, rendez-vous sur le groupe de ressources de votre chatbot via

le portail Azure, et ouvre le *App Service* créé pour le QnA. Dans la partie Overview, vous aurez accès à la tarification. Si elle est sur F1, alors très bien. Sinon, rendez-vous dans la section "Change App Service Plan" dans la barre déroulante de gauche, puis créez un nouvel *App Service Plan* à la bonne tarification. Allez ensuite supprimer celui qui n'est plus utilisé.

3) Liaison entre cette base de connaissances et le chatbot

Il va maintenant falloir modifier le code de votre bot afin que celui-ci intègre la base de connaissances :

- Première étape : installer les packages suivants (avec pip comme au lors de la création du chatbot) : botbuilder-ai et flask

- Deuxième étape : ajouter ces lignes dans votre config.py

```
QNA_KNOWLEDGEBASE_ID = os.environ.get("QnAKnowledgebaseId", "<knowledge-base-id>")
QNA_ENDPOINT_KEY = os.environ.get("QnAEndpointKey", "<qna-maker-endpoint-key>")
QNA_ENDPOINT_HOST = os.environ.get("QnAEndpointHostName", "<your-hostname>")
```

- Troisième étape : modifier le app.py en ajoutant l'argument "CONFIG" (ligne 60)

```
BOT = MyBot(CONFIG)
```

- Quatrième étape : modifier le bot.py.
  D'abord ajouter quelques imports au début du fichier (pensez également à ajouter les packages botbuilder-ai et flask dans le requirements.txt)

```
from flask import Config

from botbuilder.ai.qna import QnAMaker, QnAMakerEndpoint
from botbuilder.core import MessageFactory
```

Puis ajouter une fonction _init_ au début de la classe :

```
def __init__(self, config: Config):
   self.qna_maker = QnAMaker(
      QnAMakerEndpoint(
         knowledge_base_id=config.QNA_KNOWLEDGEBASE_ID,
```

```
            endpoint_key=config.QNA_ENDPOINT_KEY,
            host=config.QNA_ENDPOINT_HOST,
        )
    )
```

- Enfin, cinquième et dernière étape, il faut modifier légèrement la fonction on_message_activity de la manière suivante :

```
async def on_message_activity(self, turn_context: TurnContext):
    # The actual call to the QnA Maker service.
    response = await self.qna_maker.get_answers(turn_context)
    if response and len(response) > 0:
        await turn_context.send_activity(MessageFactory.text(response[0].answer))
    else:
        await turn_context.send_activity("No QnA Maker answers were found.")
```

C'est à priori dans la dernière ligne que sera effectué le code si aucune réponse n'est trouvée dans la base de connaissances.

Il ne reste maintenant plus qu'à redéployer le chatbot pour que ces changements soient effectifs.

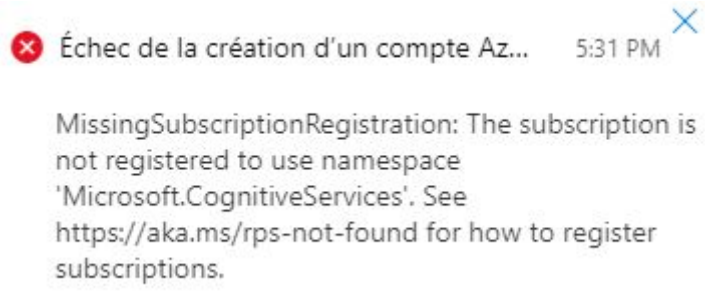
LA RECHERCHE DE FICHIERS

1) Utilisation de Luis

Dans cette partie nous allons dérouler le processus de création et d'utilisation d'une ressource Luis.

Dans un premier temps, rendez-vous sur luis.ai. Vous pourrez alors cliquer sur "nouvelle ressource de création". Choisissez alors l'abonnement, le groupe de ressource associé (celui du chatbot, créé lors du déploiement), et le nom de la ressource.

Il est probable que vous rencontriez l'erreur suivante :



Dans ce cas, rendez-vous sur le portail Azure, et ouvrez le menu en haut à gauche. Vous trouverez dans la barre déroulante l'accès aux *Subscriptions*. Cliquez sur la souscription avec laquelle vous travaillez depuis le début, puis cliquez sur *Resource Providers* dans la barre déroulante à gauche de votre écran. Filtrez ensuite sur "Microsoft.CognitiveServices", et cliquez sur *Register*. Il est probable que vous deviez cliquer plusieurs fois pour que la modification soit prise en compte. Une fois cela fait, vous pouvez retourner sur l'onglet Luis, et cliquer sur terminer.

Votre ressource ayant été créée, vous pouvez maintenant créer une nouvelle application. Ouvrez l'onglet *Gérer* en haut à droite de l'écran. Dans la barre de gauche, cliquez sur *Versions*, puis importez le fichier .lu que nous vous avons fourni. Cliquez ensuite sur *Entraîner* en haut à droite, puis sur *Publier* juste à côté. Rendez vous ensuite dans la partie *Ressources Azure* dans la barre à gauche de votre écran. Dans la partie *Ressources de Prévision*, créez une nouvelle ressource de prévision s'il n'y en a pas déjà une.

Vous pourrez ensuite récupérer l'exemple de requête fourni, puisqu'il contient des clés essentielles pour relier Luis au chatbot. Elle sera de la forme :

```
https:/<LUIS_API_HOSTNAME>/luis/prediction/v3.0/apps/<LUIS_APP_ID>/slots/production/predict?subscription-key=<LUIS_API_KEY>&verbose=true&show-all-intents=true&log=true&query=YOUR_QUERY_HERE
```

Rendez vous dans le fichier config.py du chatbot, puis mettez à jour les valeurs suivantes avec celles que vous venez d'obtenir.

```
LUIS_APP_ID = os.environ.get("LuisAppId", "<LUIS_APP_ID>")
LUIS_API_KEY = os.environ.get("LuisAPIKey", "<LUIS_API_KEY>")
# LUIS endpoint host name, ie "westus.api.cognitive.microsoft.com"
LUIS_API_HOST_NAME = os.environ.get("LuisAPIHostName", "<LUIS_API_HOST_NAME>")
```

Finalement, re-déployez votre chatbot, et vous pourrez alors utiliser la fonctionnalité de Luis.

2) Recherche dans les répertoires partagés Windows

Pour le moment, la recherche de fichiers Windows ne se fait que sur les fichiers locaux à l'archive envoyée lors du déploiement. Si le projet est repris, il suffirait alors de programmer l'accès à une base de dossiers partagés Windows, et le reste du code serait toujours fonctionnel.

3) Rechercher dans SharePoint

L'accès à la base SharePoint se fait à l'aide de son URL, et d'un compte utilisateur sur le domaine de Figeac Aero, avec les accès associés. Il doit impérativement avoir été désactivé la double authentification pour ce compte, car cela bloque l'accès depuis la fonction Python sinon.

Si le compte d'accès ou le lien du SharePoint souhaite être modifié, il faudra alors se rendre dans le fichier config.py, et actualiser la partie ci-dessous.

```
SHAREPOINT_USERNAME = os.environ.get("SharapointId","<email-compte-service>")
SHAREPOINT_PASSWORD = os.environ.get("SharepointPassword","<password>")
SHAREPOINT_SERVER_URL = os.environ.get("SharepointServerUrl","https://fga01.sharepoint.com/sites/DSI-FigeacAero-ProjetChatBotavecTelecomNancy")
```

Il est important de noter que pour le moment, le chatbot ne différencie pas les droits d'accès de l'utilisateur des siens, et par conséquent peut renvoyer des fichiers auxquels l'utilisateur ne devrait pas avoir accès.

4) Recherche dans Teams

La recherche dans les fichiers Teams fonctionnant de la même manière que dans un SharePoint classique, il faudra alors seulement récupérer l'adresse du SharePoint associé au Teams en question, et le fournir dans le fichier config.py.

```
TEAMS_SP_URL = os.environ.get("TeamsSharepointUrl","<URL>")
```

L'URL de ce SharePoint peut être trouvée en ouvrant un fichier dans le Teams voulu, puis en cliquant sur les … en haut à droite puis sur "Ouvrir dans SharePoint".

L'INTÉGRATION DANS MICROSOFT TEAMS

Enfin, l'intégration du bot final dans l'application Microsoft Teams est très simple : il vous suffira de récupérer l'identification de l'application Microsoft Azure créée (par exemple accessible via votre groupe de ressource > votre Bot Channels > Registration > Configuration > Microsoft App Id), puis de la coller dans les champs *id* et *botId* du fichier teams_app_manifest/manifest.json. Vous pourrez également y changer le nom sous lequel le bot sera accessible aux utilisateurs, sa description, etc… Vous pourrez ensuite sélectionner les deux fichiers de l'icône et le manifest pour les compresser au format .ZIP.

Ensuite, il vous faudra vous rendre sur le Teams voulu (ici celui de Figeac Aero) avec les droits administrateur. Vous pourrez alors accéder à l'onglet Applications en bas à gauche, puis à la ligne "Charger une application personnalisée" tout en bas à gauche dans la barre déroulante. Vous pourrez y déposer votre archive fraîchement créée, puis l'activer en cliquant sur son encadré dans la fenêtre qui vient de s'actualiser. Désormais, toutes les personnes ayant accès à ce Teams pourront accéder au chatbot dans leurs applications, et pourront choisir de l'épingler à leur menu de gauche.