Projet ChatBot - ProgRes 2017

Le but du projet est de créer un ChatBot qui s'intègre dans l'application Discord. Le ChatBot devra répondre à plusieurs type de messages pour permettre de simplifier certaines taches (pas forcément très utiles) aux participants d'une conversation. Le ChatBot utilisera plusieurs API pour fonctionner.

1. Mise en place de votre Bot

Tout d'abord, créez un compte sur Discord (https://discordapp.com/) et créez une application sur https://discordapp.com/developers/applications/me. Convertissez votre App en Bot pour obtenir un token.

Après avoir rejoins le serveur PROGRES-UPMC, ajoutez votre bot au serveur à l'aide du lien https://discordapp.com/api/oauth2/authorize?client_id=LeClientIdDeVotreApp&scope=bot&permissions=0 (en replaçant `LeClientIdDeVotreApp` par le Client Id de votre application).

Pour simplifier l'utilisation de l'API Discord, nous utiliserons le package NodeJs discord.js : https://discord.js.org

Commencez par créer un nouveau projet NodeJs et installez `discord.js` (**Toutes les erreurs du type `UNMET PEER DEPENDENCY` pourront être ignorées**):

```
npm install discord.js --save
```

```
const Discord = require('discord.js');
const client = new Discord.Client();

client.on('ready', () => {
  console.log('I am ready!');
});

client.on('message', message => {
  if (message.content === 'ping') {
    message.reply('pong');
  }
  console.log(message);
});

client.on('presenceUpdate', function(oldMember, newMember) {
  console.log(oldMember.presence, '=>', newMember.presence);
});

client.login(process.env.DISCORD_TOKEN);
```

L'utilisation de l'api de `discord.js` est similaire à celle de l'API Slack mais possède des différences.

L'événement `message` ne capture que les nouveaux messages (contrairement à Slack où tous les événements était considérés comme des messages). Les messages reçus possèdent beaucoup d'attributs (`content` pour le contenu par exemple) mais aussi des méthodes (`reply` pour répondre directement par exemple). Liste des attributs : https://discord.js.org/#/docs/main/stable/class/Message.

Il existe beaucoup d'autres événéments (par exemple `messageUpdate` lorsqu'un message est modifié): liste complète des événements: https://discord.js.org/#/docs/main/stable/class/Client

Dans l'exemple donné ci-dessus, l'événement `presenceUpdate` est écouté, permettant d'être averti lorsqu'un utilisateur se connecte ou se déconnecte.

2. Fonctionnement de votre Bot

Votre Bot doit toujours répondre quelque chose lorsque quelqu'un lui parle directement (direct message) ou lorsque quelqu'un le tag. De plus, Il doit envoyer un message à l'utilisateur `bramas` lorsque l'utilisateur `bramas` se connecte. Ce message doit être de la forme : "Bonjour maitre, je suis le bot de ..... et de ....., que puis-je faire pour vous aujourd'hui?" (ou les points .... sont remplacés par le nom des membres du binôme).

Votre bot ne doit jamais envoyer de messages autres que ceux cités au-dessus.

Lorsqu'un message est adressé au Bot, si le message contient une requête comprehensible par votre Bot, alors il doit y répondre en fonction, sinon il doit répondre un message pour dirr qu'il n'a pas compris.

2.1. Requêtes par commandes

Votre Bot doit pouvoir répondre à plusieurs types de requêtes. La première manière de détecter une requête et de supposer que le message reçu contient un texte prédéfini pour chaque type de requête. On pourra supposer par exemple que les messages envoyés à votre Bot doivent être de la forme "!nom_de_la_commande" (c'est a dire un point d'exclammation suivi d'un mot représentant la commande)

Les sous-sections suivantes décrivent plusieurs requêtes que votre Bot peut prendre en charge mais vous n'êtes pas obligés de toutes les implémentées, et vous pouvez en choisir d'autres. La note finale dependra du nombre et de la complexité des requêtes que votre bot supporte, mais aussi du fonctionnement globale de votre bot, du code source, du rapport et de la vidéo.

2.1.1 Raconter une blague

Si votre Bot reçoit un message contenant `"!blague"`, alors il doit utiliser l'API du site https://www.chucknorrisfacts.fr pour répondre par une blague aléatoire.

2.1.2 Affichage de la météo

Si votre Bot reçoit un message contenant `"!météo"` suivi d'une ville, alors il doit utiliser l'API du site http://openweathermap.org/api pour répondre par les prévisions météo du lendemain de la ville indiquée.

2.1.3 Recherche d'image

Si votre Bot reçoit un message contenant `"!image"` suivi d'un mot, il doit envoyer une image en lien avec le mot entré.

Utilisez l'API d'Imgur (https://api.imgur.com/endpoints/gallery#gallery-search) pour rechercher une image en liens avec le mot entré.

2.1.4 Affichage de la position de l'ISS sur une carte

Si votre Bot reçoit un message contenant `"!iss"`, il doit envoyer une image (url de l'image dans un objet embed) contenant la position de l'ISS sur une carte.

Utilisez l'api de http://wheretheiss.at pour récupérer les coordonnées GPS de l'ISS (url : `https://api.wheretheiss.at/v1/satellites/25544`)

Utilisez l'API d'openstreetmap pour récupérer l'image correspondant à des coordonnées GPS données (remplacer `LAT` et `LNG` par la latitude et la longitude) : `http://staticmap.openstreetmap.de/staticmap.php?center=LAT,LNG&zoom=5&size=400x300&maptype=mapnik&markers=LAT,LNG,ltblu-pushpin`

Une extension intéressante consiste à replacer le pointeur inséré automatiquement, par une image représantant un satellite. Pour cela il faut télécharger la carte ou se trouve l'ISS, insérer l'image du satellite (à l'aide du package `sharp` par exemple) puis, envoyer l'image obtenue sur le channel.

2.2. Requêtes en langage naturel.

Un manière de rendre votre bot plus accessible est de le munire de compréhension du langage naturel. Pour cela vous pouvez utiliser l'api `wit.ai` permettant de convertir un message reçu en un ensemble de commandes predefinies.

3. Déploiement

Le code source de votre Bot doit être hébergé sur Github et être déployé sur Heroku.

Pour que l'application fonctionne sans arrêt sur heroku, il faut qu'elle contienne un serveur web. Le plus simple est donc d'ajouter un exemple d'application `express` dans votre fichier principal :

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
  res.send('hello world');
});
app.listen(process.env.PORT || 5000);
```