TUTORIEL ROBOT (BOT) POUR *DISCORD* EN *JAVASCRIPT*

Version 3.2

DIVERSITY by EPITECH

Sommaire

Consignes

* Pour ce projet-là, il vous sera demandé de créer un repository avec le nom : *cc_bot_discord_javascript*.
* N'oubliez pas de push régulièrement !
* En cas de question, pensez à demander de l'aide à votre voisin de droite. Puis de gauche. Ou inversement. Puis demandez enfin à un Cobra si vous êtes toujours bloqué(e).
* Pensez à faire valider chaque partie que vous réaliserez à un Cobra lorsque vous l'aurez terminée.
* N'hésitez pas à faire des bonus et à ajouter des fonctionnalités lorsque votre projet sera terminé et validé.
* Vous avez tout à fait le droit d'utiliser internet pour trouver des réponses ou pour vous renseigner.

Présentation

L'été arrive bientôt et la directrice camping de Pokéville aimerait utiliser cette année quelque chose qui a beaucoup aidé les habitants de la ville l'hiver dernier : un bot Discord. La directrice souhaiterait divers aménagements en prévision avec les différentes activités du camping mais aussi sur l'aspect sécurité mais ne s'y connait pas du tout en programmation…

Ayant discuté en avance avec Monseigneur Node de ce projet, ce dernier souhaitait que ce soit vous, son disciple, qui ayez la charge de cette tâche !

Une condition particulière que la directrice du camping aimerait que vous suiviez, c'est que le bot soit à l'effigie de son cher petit *Carapuce* !

Ici vous n'utiliserez seulement qu'un fragment de toutes les possibilités qu'offre le développement de bot Discord !

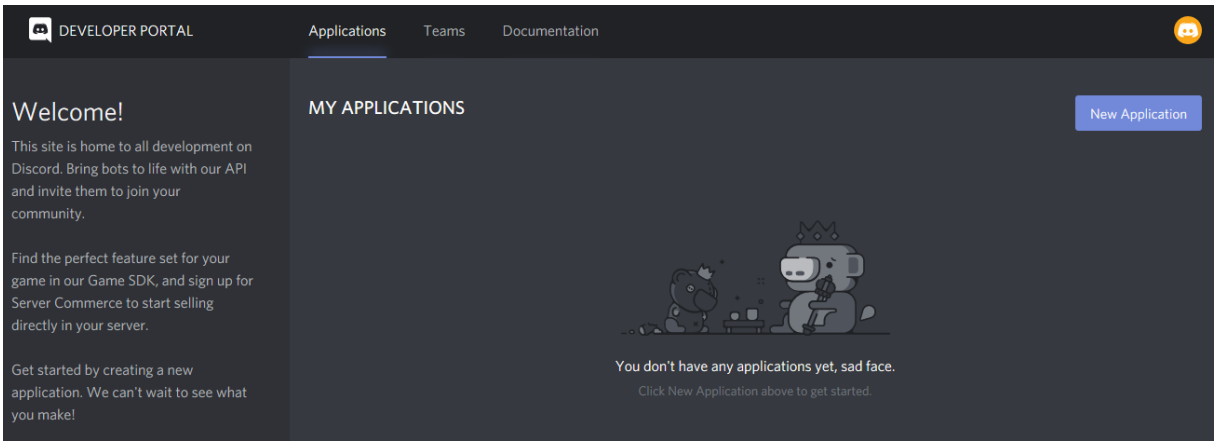
Vous pourrez trouver toute la documentation pour votre bot en JavaScript sur ce site (onglet **Documentation**).

I - Création du Bot *Carapuce*

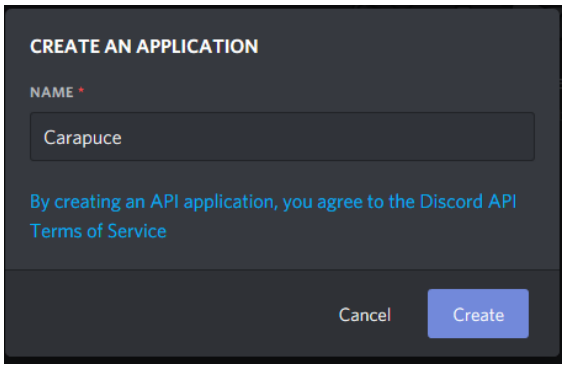
A - Prends vie !

Pour commencer, vous allez créer un bot en allant sur la page du site officiel de *Discord* dédié à cet effet.

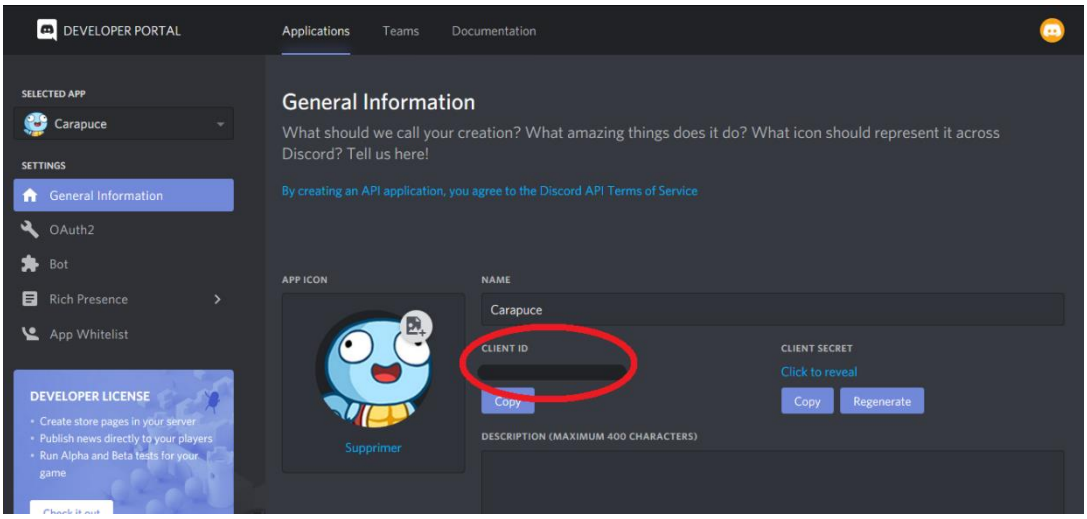
Maintenant identifiez-vous avec votre compte *Discord*. Si vous n'avez pas de compte *Discord*, créez-en un.



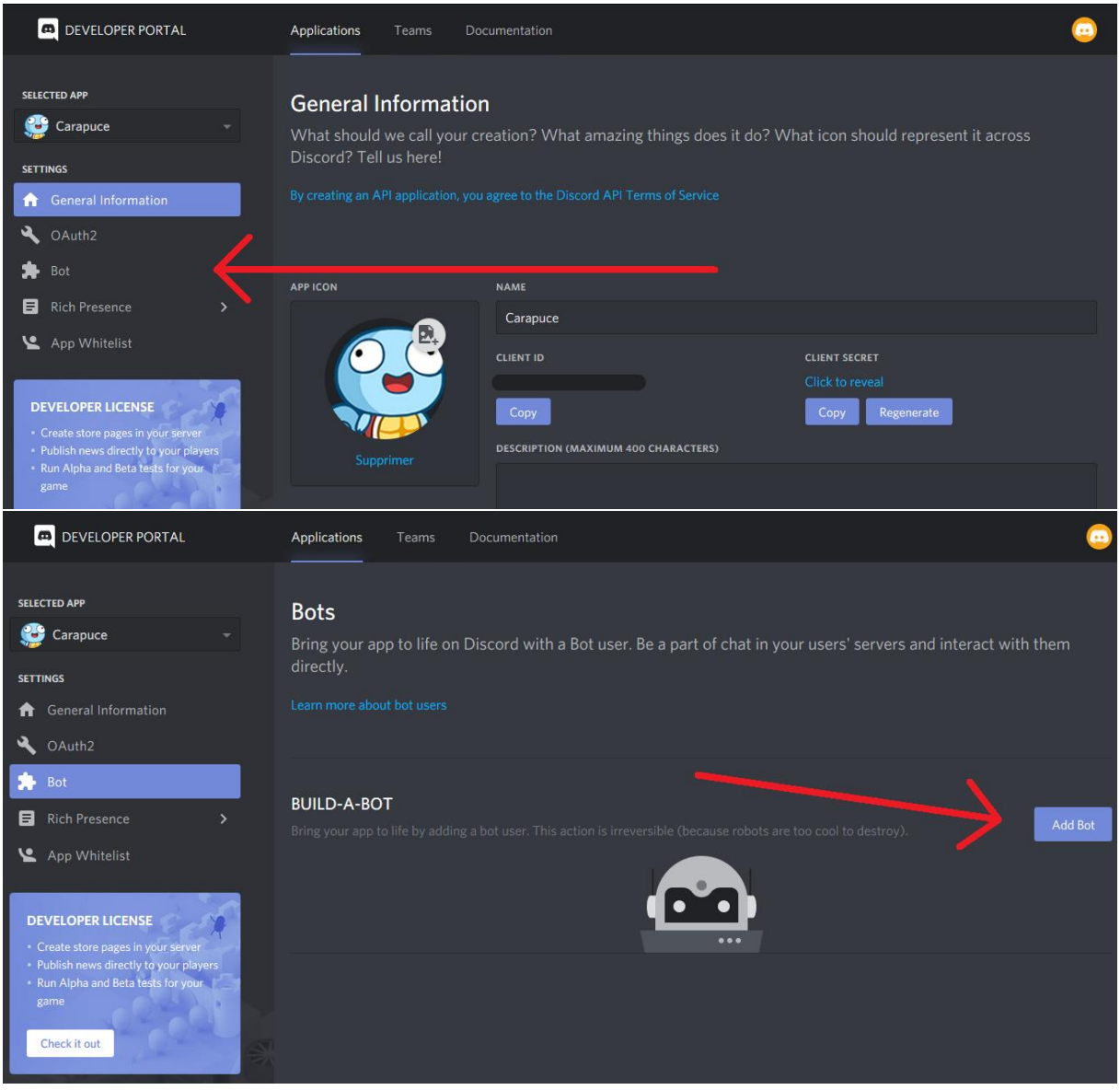
Ensuite, cliquez sur **New Application** et donnez un nom au bot (vraiment au hasard : *Carapuce*).



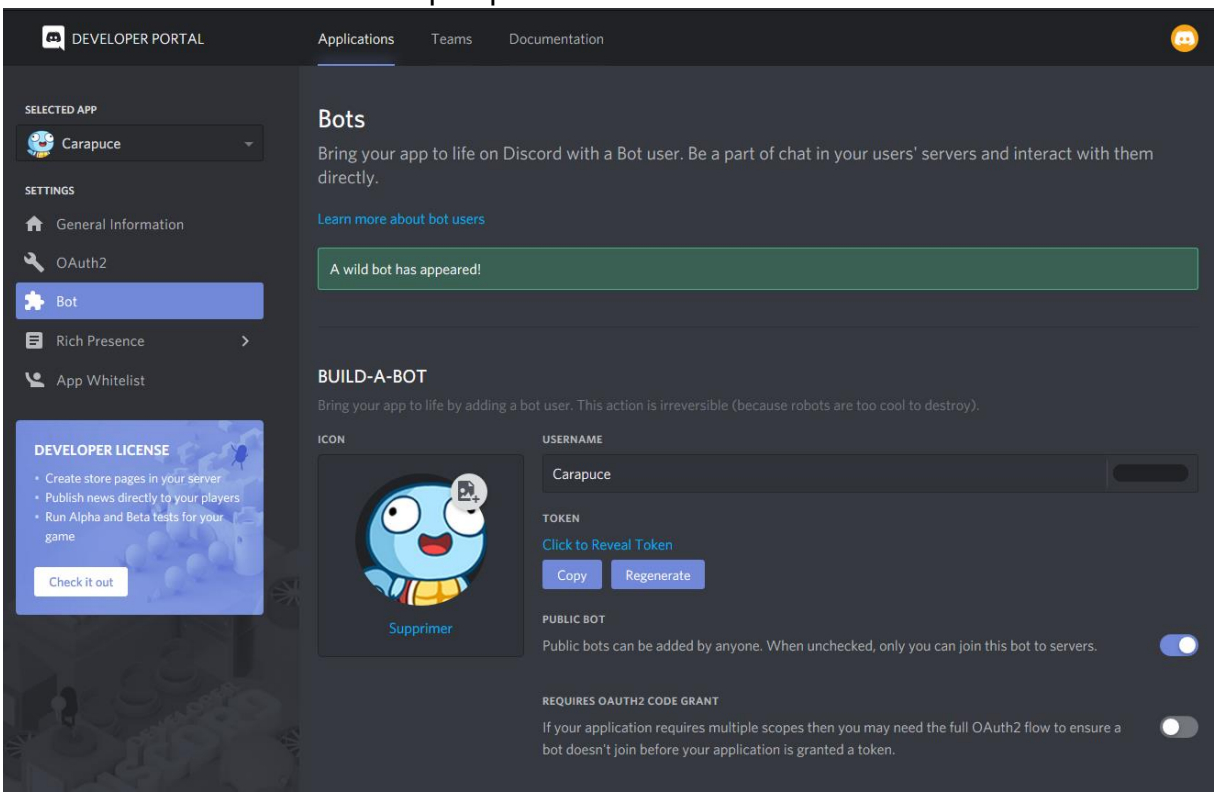
Afin de lui donner un semblant de vie supplémentaire, donnez-lui une image de profile.



Sur la liste de menus sur la gauche cliquez sur **Bot**, puis **Add bot.**

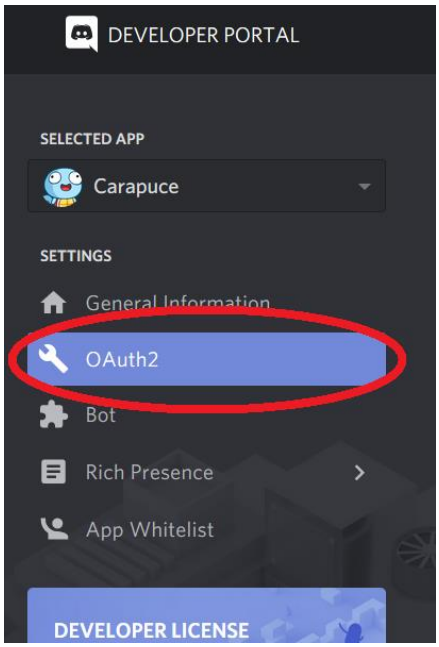


Vous obtiendrez alors quelque chose de similaire à cela :

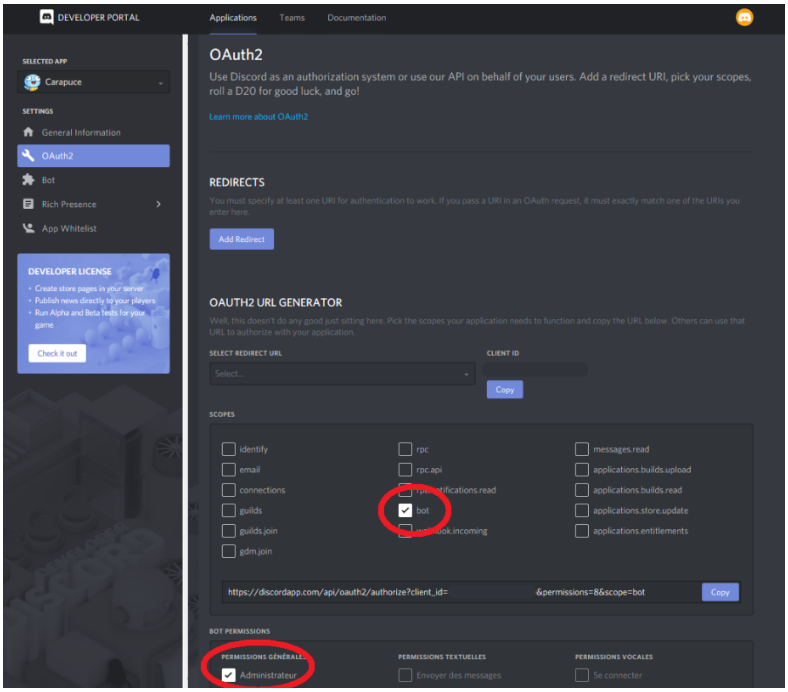


B - Viens sur le serv' !

Maintenant il va falloir inviter votre bot sur le serveur Discord du camping de Pokéville ! Pour commencer, allez sur l'onglet « *OAuth2* » (à gauche).



Cochez la case **Bot**. Puis, plus bas, Cochez la case **Administrateur** (cocher cette case signifie cocher toutes les autres implicitement).



Maintenant que le lien d'invitation de votre bot est accessible, cliquez dessus ! Sélectionnez le serveur sur lequel vous voulez inviter votre bot (je vous conseille de l'inviter sur un serveur à part afin de faire votre configuration de bot tranquillement).

Félicitations, vous avez atteint le niveau 1 !

Vous avez réussi à créer un bot et à le mettre sur un serveur.

II - Mise en place du Bot *Carapuce*

Tout d'abord, il vous faudra suivre le document d'installation de NodeJS ainsi que les spécificités pour Discord.

Un petit outil en plus pour laisser votre bot tourner et pouvoir utiliser votre terminal en même temps : **pm2**.

Les informations pour son installation et son utilisation dans le document d'installation de NodeJS et ses outils.

Pour le bon déroulé de votre travail, il vous faudra installer *FFMPEG* pour jouer de la musique. Pour cela, vous pourrez trouver les informations nécessaires dans le document d'installation de FFMPEG.

Félicitations, vous avez atteint le niveau 2 !

Vous avez fait toutes les installations pour le bot.

III - Mon premier échange avec le Bot Carapuce

Pour commencer, il va falloir créer un fichier Bot.js (.js étant, on le rappelle, l'extension pour le JavaScript).

Le fichier que vous devez créer doit se trouver dans le dossier (représentant le repository Git) que vous avez préalablement créé.

Dès lors, il vous faudra obligatoirement ce bout de code dans votre fichier !

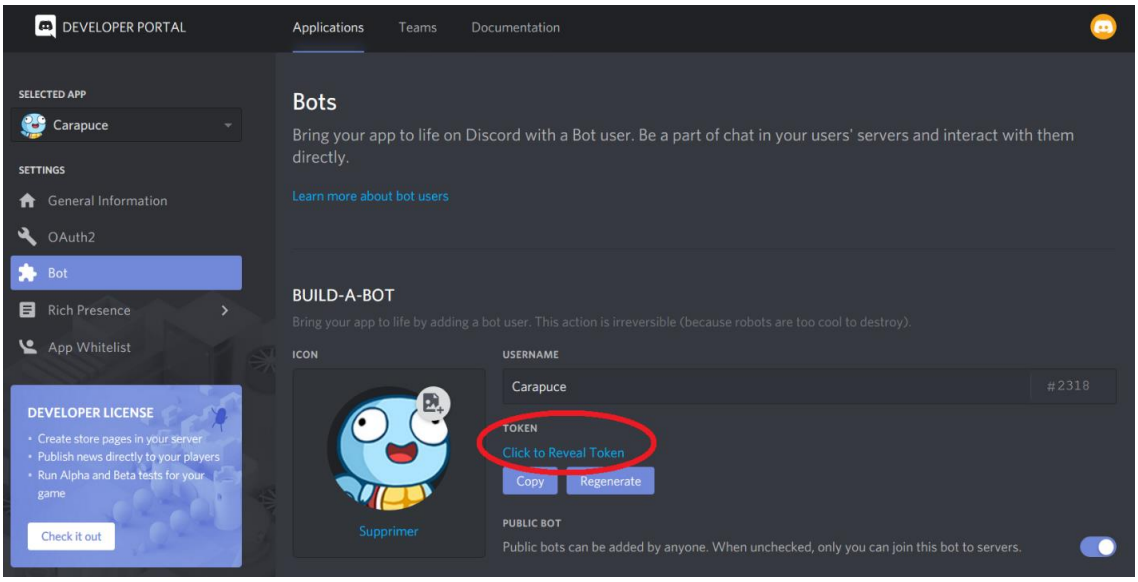
```
//ici on précise qu'on aura besoin du module discord.js qui lui appelle l'API de Discord
const Discord = require("discord.js");
//ici on appelle une fonction de l'objet créé au-dessus qui permet de récupérer un objet utilisateur client qui représente le bot
const bot = new Discord.Client();

bot.on("ready", function () {
    //ici on affichera dans le terminal que le bot est bien connecté
    console.log("Carapuce est dans les places !");
});
//la ligne suivante permet d'indiquer à l'objet Discord qui est notre bot afin qu'il puisse se connecter
bot.login("MON_TOKEN");
```

/!\ Ne divulguez pas votre token /!\

Une personne mal intentionnée pourrait faire n'importe quoi sur les serveurs sur lesquels se trouve votre bot !

MON TOKEN sera à remplacer par le celui du bot. Pour l'obtenir, il vous faudra retourner sur la page de développement de bot *Discord*, dans l'onglet **Bot** puis vous verrez en-dessous de **Username**.



Maintenant essayez d'avoir une première interaction avec *Carapuce* !

Pour cela *Carapuce* va devoir regarder les messages envoyés sur les salons textuels et effectuer une action spécifique en fonction du contenu des messages.

Pour faire cela, ajoutez à votre script ce bout de code :

```
//ici on regarde quand le bot est en ligne et qu'il voit passer un message (peu importe le serveur)

bot.on("message", message => {
    //on regarde alors si le contenu du message est strictement égale à "!ping"
    if (message.content === "!ping") {
      //si oui on arrive ici et on envoie, dans le channel d'où provient le message, "Carapong !"
      message.channel.send("Carapong !");
    }
});
```

Maintenant il faut tester pour vous assurer de si ça fonctionne ou non. Pour cela, depuis le terminal de commande **cmd** rentrez la commande **node Bot.js** ou **pm2 start Bot.js** si vous avez installé *pm2*.

Allez sur le serveur et dans un salon textuel, tapez **!ping**.

Et ça fonctionne, félicitations ! Mais ce n'est pas tout ce que vous pouvez faire avec les messages, nous verrons d'autres utilisations dans la partie suivante.

Félicitations, vous avez atteint le niveau 3 !

Vous avez fait toutes les installations pour le bot.

IV - Décorticage d'un message

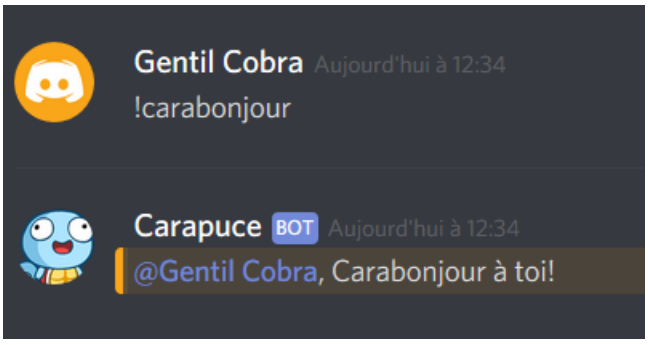
D'après la documentation sur les messages, un message, qui est un objet, a plein de propriétés (qui sont, elles-mêmes, des objets) qui permettent de connaître :

* Qui est l'auteur du message
* Si le message a été supprimé, épinglé…
* Les réactions à ce message
* Etc…

Mais il dispose aussi de méthodes (entendez par là des fonctions qui lui sont propres). Certaines permettent d'épingler le message, d'y répondre ou même d'y réagir !

Dans un premier temps, et si vous essayiez de répondre à un message ? Pour cela, essayez de rentrer le code suivant à la suite du dernier.

Si vous pensez avoir trouvé la bonne réponse, relancez votre bot et si vous obtenez ceci, vous aurez tout fait comme il faut ! Sinon, reregardez bien la documentation.



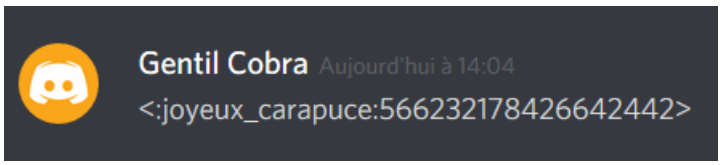
La directrice du camping aimerait que lorsque quelqu'un salut *Carapuce*, celui-ci ajoute une réaction au message de salutations.

Pour commencer simplement, il nous faudra récupérer l'emoji smiley (😄). Merci les développeurs de *Discord*, il y a une fonctionnalité qui le permet ! Pour cela, rentrez **\:smiley:** dans le tchat, et au lieu d'écrire son emoji smiley, *Discord* affichera l'emoji smiley universel.

Maintenant à votre avis, quelle méthode de l'objet message devrions nous utiliser pour réagir au message ?

Si vous avez dit **.react()** c'était la bonne réponse ! Essayez de le faire tout seul pour voir si vous y arrivez ! Sinon regardez comment faire juste après. 😉

En effet, cette méthode nous permet de réagir au message. Grâce à un emoji sous forme de phrase (exemple : ''😄'') ou sinon grâce à l'ID d'un emoji personnalisé par exemple. Pour l'obtenir, c'est de la même manière que pour obtenir l'emoji smiley. Si pour appeler votre emoji vous devez entrer **:joyeux_carapuce:** il suffira d'ajouter **** devant pour obtenir l'**identifier** (prononcez *identifailleur*) de l'emoji.

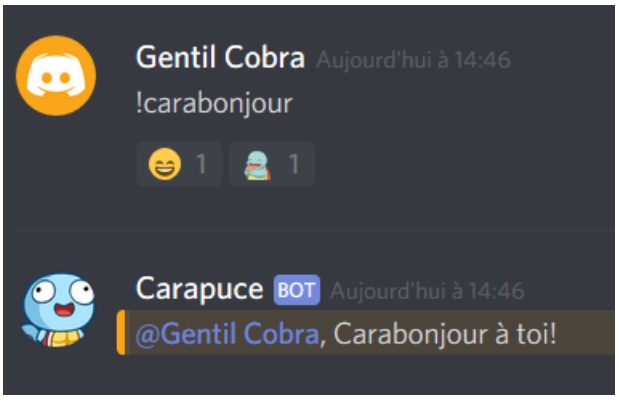


Celui-ci sera en deux parties, le nom de l'emoji et un nombre qui est son **ID**, ici juste l'**ID** qui nous intéresse. Ce dernier sera à mettre en paramètre d'une méthode qui sera elle-même le paramètre de la méthode. Exemple : **bot.emojis.cache.get(''01234567890123456 7'')**.
Si vous ne savez pas comment ajouter des emojis personnalisés sur votre serveur, ça vous est expliqué dans la partie Emojiax le maître des emojis.

Chose importante à noter, un bot ne peut pas utiliser des emojis personnalisés d'un serveur s'il n'est pas dessus !

```
if (message.content === "!carabonjour") {
    message.reply("carabonjour à toi !");
    message.react("😄");
    message.react(bot.emojis.cache.get("566232178466424 42"));
}
```

Votre fonction précédente sera à compléter de cette manière. N'oubliez pas de relancer votre bot pour qu'il puisse maintenant réagir. Vous devriez obtenir un résultat similaire à celui-ci (évidemment vous pouvez choisir un autre emoji que le smiley, et l'ID de votre emoji personnalisé ne sera pas le même que celui de l'exemple).

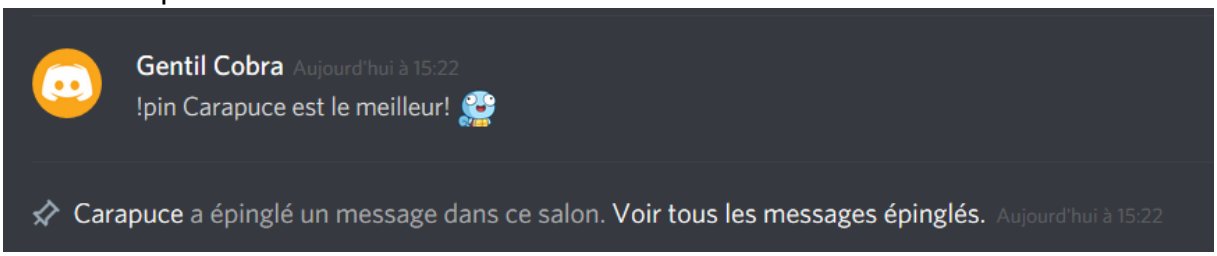


Essayez une nouvelle chose comme épingler un message. Pour cela, regardez le début du message et non le message en entier. Petite aide pour regarder le début du message, il faut utiliser une méthode de la variable objet **content** (contenu en français) de l'objet message.

Pas d'idée ? Bon, et bien pas grave. Il faut savoir que la variable objet **content** de l'objet message possède la méthode **startsWith()** (à traduire « *commence avec* ») qui prend en paramètre une expression et qui renverra vrai ou faux (à savoir, en anglais, « *true* » ou « *false* ») si le message commence par l'expression passée en argument. Ce qui en termes de code serait égale à :

```
if (message.content.startsWith("!pin")) {
    message.pin();
}
```

Ce qui devrait vous donner comme rendu ceci :



Maintenant que vous avez fait ça, vous avez déjà vu pas mal de choses intéressantes à faire avec les messages des utilisateurs et qui vous aideront pour les étapes suivantes. 😉

Félicitations, vous avez atteint le niveau 4 !

Vous savez maintenant utiliser et interpréter, en partie, les messages.

V - Les embeds ça embellit

Le carapuce de la directrice du camping adore vous regarder coder et voir le projet prendre forme petit à petit ! 😉

La directrice du camping étant souvent sur plusieurs fronts a du mal à se souvenir des différentes commandes auxquelles elle a accès. Elle vous demande donc si vous pourriez lui faire un pense-bête ? Quoi de mieux qu'un ***embed*** dans ce cas ?

Si vous utilisez déjà *Discord* et que vous êtes sur un serveur sur lequel il y a un bot, vous avez très certainement remarqué que vous pouvez demander au bot de vous dire quelles commandes vous pouvez utiliser avec lui. C'est ce que vous allez faire maintenant !

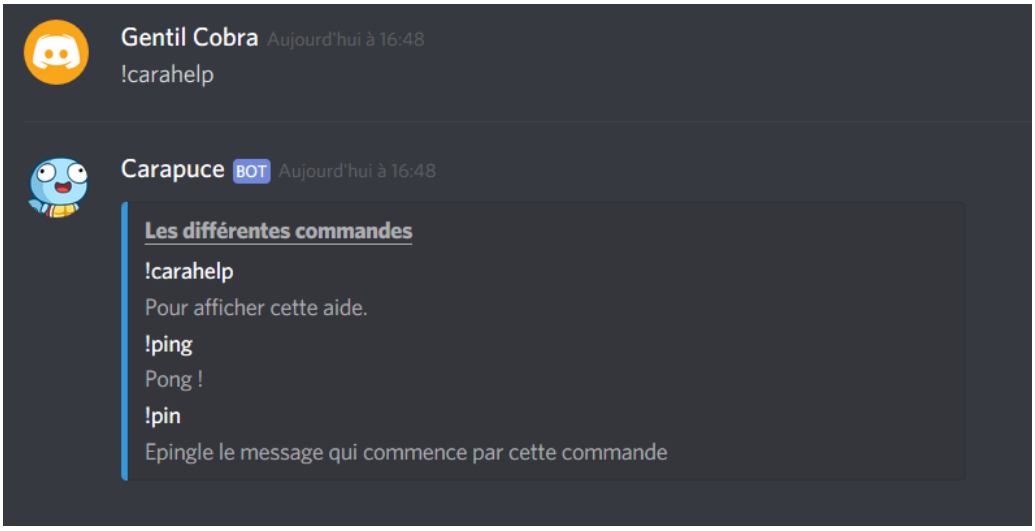
Attention cependant, il existe plusieurs façons de faire des embeds avec la bibliothèque discord.js.

Pour commencer, il va falloir regarder si le contenu du message est !carahelp. Et nous renverrez un message dans le salon textuel mais le message envoyé sera de type *embed*.

Pour ça, vous **pouvez** faire de la manière suivante :

```
if (message.content === "!carahelp") {
    //on envoie un message de type embed dans le channel d'où
provient le message
    message.channel.send({
      embed: {
            color: 3447003,
            description: "__**Les différentes commandes**__",
            fields: [
                  {
                        name: "!carahelp",
                        value: "Pour afficher cette aide."
                  },
                  {
                        name: "!ping",
                        value: "Pong !"
                  }
            ]
      }
    });
}
```

Le résultat sera alors le suivant :



Pensez à le mettre à jour au fur et à mesure de votre avancée pour être sûr de ne rien oublier. Ça serait vraiment gentil pour la directrice du camping. 😉

Félicitations, vous avez atteint le niveau 5 !

Vous savez maintenant utiliser des embeds.

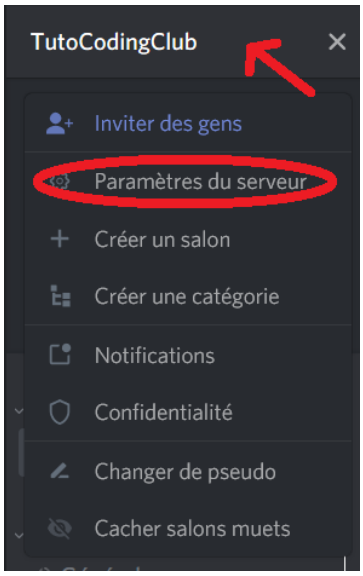
VI - Emojiax le maître des emojis

Tout d'abord voici comment mettre des emojis personnalisés sur un serveur !

Pour commencer, il vous faut des images (ce qui parait logique). Conseil, utilisez des emojis comme ceux de *Twitch* (c'est à dire détourés, comme la tête de **Kappa** par exemple) de prendre des images en PNG pour avoir le fond transparent.

Sinon prenez des images de *Carapuce* (pour faire plaisir à la directrice du camping). 😉

Maintenant, allez dans les paramètres du serveur.



Puis cliquez sur le menu **emoji** puis **Ajouter un emoji**.

Il ne vous reste plus qu'à choisir l'image qui servira d'emoji. Puis recommencez pour chaque emoji que vous souhaiterez ajouter. Pour information, vous pouvez choisir de renommer l'emoji. Par exemple, si vous devez écrire **:carapuce:** pour utiliser un de vos emojis personnalisés, vous pouvez redéfinir pour que vous ayez à écrire **:squirtle:** à la place.

La directrice du camping de Pokéville souhaiterait que ses campeurs puissent voir les différents emojis du serveur, facilement. Donc que lorsque quelqu'un envoie la commande !emojiliste, que *Carapuce* renvoie chaque emoji personnalisés du serveur.

Voilà le résultat attendu :

Si vous ne savez pas du tout comment vous y prendre, ou que vous avez essayé quelque chose mais que ça ne marche pas, essayez de faire avec ce code-là :

```
if (message.content === "!emojiliste") {
    const emojiliste = message.guild.emojis.cache.map((e) => e);
    message.channel.send(emojiliste);
}
```

Cependant, les autres utilisateurs ne savent pas ce qu'ils doivent écrire pour utiliser l'emoji.

Vous pouvez récupérer cette information en regardant parmi les propriétés de l'objet **e**, ici. L'objet **e**, ici, est un objet de type *Emoji,* vous trouverez sûrement cette information sur la documentation dans l'onglet approprié. 😉

Il faut savoir que dans une majorité des langages haut niveau, comme le C++, le JavaScript et le Python, il est possible de « concaténer » des phrases entre elles. C'est-à-dire de la coller bout à bout. Exemple : ''Hello'' + ''World'' donnerait ''HelloWorld''.

Petite astuce de plus, vous devez faire cette opération dans les parenthèses de **message.guild.emojis.cache.map()** !

Le but étant d'obtenir ce résultat :

Carapuce voudrait que le bot puisse réagir avec un emoji personnalisé quand le message contient un mot particulier (« carapuce » par exemple). Et surtout il veut que ça fonctionne à tous les coups !

Précision : Il faut que ça fonctionne sans prendre compte les majuscules et minuscules. Que le message contienne « carapuce », « Carapuce » ou même « cArAPucE ».

Essayez de trouver comment faire, ce n’est en soit pas compliqué. 😉

/!\ Attention /!\
Lorsque vous utiliserez des emojis personnalisés, s’ils sont supprimés, votre code risque d’échouer et de planter !

Félicitations, vous avez atteint le niveau 6 !

Vous avez réussi à gérer des réactions sur les messages.

VII - Le Cara-quiz

La directrice a regardé les prévisions météo et une de ses activités du soir risque de devoir être annulée car elle ne pourra être faite s'il fait mauvais temps… Elle aimerait donc voir avec vous si une alternative via Discord est envisageable ? En effet il s'agissait d'un quiz en plein air. Mais vous la rassurez et lui indiquez que c'est possible de faire cette activité sur Discord. Le bot devra alors poser une question ainsi que proposez les réponses, le tout sous forme d'**embed** parce que c'est plus joli. L'utilisateur n'aura qu'à saisir la lettre correspondant à sa réponse.

Le résultat doit être similaire au suivant :

La directrice n'ayant pas encore prévu quelles questions seront posées, elle vous laisse vous occuper d'un exemple en attendant de les trouver.

Félicitations, vous avez atteint le niveau 7 !

Vous avez sauvé des soirées du camping pour l’été prochain.

VIII - Un peu de manipulation d'images

Ce que vous la directrice souhaite ici, c'est de créer une image personnalisée pour accueillir les campeurs lorsqu'ils arrivent sur le serveur.

La première chose à faire, c'est d'ajouter la ligne suivante en haut de votre fichier, avec les autres déclarations de dépendances.

```
const Canvas = require("canvas");
```

Pour pouvoir faire quelque chose lorsque quelqu'un rejoint un serveur, vous aurez besoin de cette ligne de code qui, si vous vous souvenez, ressemble à la ligne permettant d'intercepter les messages.

```
bot.on("guildMemberAdd", async member => {});
```

La prochaine étape sera de récupérer le salon textuel d'arrivée des utilisateurs et en même temps vérifier que ce salon existe. Si le salon n'est pas défini vous ne pourrez pas envoyer de message (ce qui tombe sous le sens) donc il faudra quitter la fonction.

Ensuite, créez un canvas (ou « canevas », « toile » en bon français) qui sera le support pour manipuler les images. Par la même occasion vous allez définir que vous utiliserez une image en 2D, charger l'image et la mettre sur le canvas.

Pour cela, vous pouvez vous y prendre de la manière suivante :

```
//on crée un support pour notre image aux dimensions 1024 x 700
   const canvas = Canvas.createCanvas(1024, 700)
   //on précise qu'on veut faire de la 2D
   const ctx = canvas.getContext('2d')
   //on charge une image qui sera notre fond
   const background = await Canvas.loadImage("./welcome.png");
   //on applique l'image sur notre support
   ctx.drawImage(background, 0, 0, canvas.width, canvas.height);
```

Carapuce souhaiterait que vous affichiez le nom du nouvel utilisateur sur l'image. Pour cela vous allez faire une petite fonction à part qui va redimensionner la taille de la police pour pas que le nom de l'utilisateur soit trop gros.

Pour cela, vous pouvez faire de la manière suivante :

```
const applyText = (canvas, text) => {
    const ctx = canvas.getContext('2d');
    //on déclare la taille de votre police d'écriture
    let fontSize = 70;

    do {
        //on affecte la taille de la police du contexte et on
décrémente la taille pour pouvoir mesurer à nouveau
        ctx.font = `${fontSize -= 10}px sans-serif`;
        //on compare la largeur en pixels du texte sur le canvas
moins la taille approximative de l'avatar
    } while (ctx.measureText(text).width > canvas.width - 300);
    //on renvoie le résultat qu'on pourra alors utiliser
    return ctx.font;
};
```

Maintenant il faut appeler la fonction précédente afin qu'elle fasse son boulot, puis définir la couleur du texte et finalement remplir la variable qui doit contenir le texte avec le texte en question.

Vous pourriez par exemple faire de la manière suivante :

```
ctx.fillText(member.displayName, 20, 685)
```

Maintenant vous allez ajouter l'avatar puis envoyer l'image finale dans un message.

Si vous ne savez pas comment vous y prendre, vous pouvez faire comme suivant :

```
    ctx.beginPath();
    //ici on va faire un cercle qu'on met en haut à droite
    ctx.arc(825, 175, 125, 0, Math.PI * 2, true);
    ctx.closePath();
    ctx.clip();
    //on récupère l'image de l'utilisateur
    const avatar = await
Canvas.loadImage(member.user.displayAvatarURL);
    //on applique l'image dans le cercle
    ctx.drawImage(avatar, 700, 50, 256, 256);
    //on met le tout dans un fichier attaché pour l'ajouter au
message qu'on enverra dans le channel
    const attachment = new
Discord.MessageAttachment(canvas.toBuffer(), "welcome-image.png");
    //on envoie le message
    channel.send(`Bienvenue sur ce CaraServeur, <@${member.id}> !
<:happy_carapuce:553490319103098883>`, attachment)
});
```

Évidemment, faites un **try catch** afin de vous assurer qu'aucune erreur ne survienne et fasse planter votre bot. Cependant, faites le *try* après avoir trouvé le salon d'arrivée des nouveaux utilisateurs.

Afin de pouvoir tester ce que vous avez fait, vous pouvez simuler l'arrivée d'un nouvel utilisateur sur le serveur (vous en l'occurrence).

Pour cela ajoutez juste ce bout de code parmi les commandes que vous devez traiter :

```
if (message.content === "!carajoin") {
    bot.emit("guildMemberAdd", message.member);
    return;
}
```


Félicitations, vous avez atteint le niveau 8 !

Vous avez réussi à faire de la manipulation d'images et elles sont belles !

IX - Le Cara-DJ est dans la place !

Une des activités phares des campings ce sont les karaokés ! Toujours dans le doute de ne pouvoir faire cette activité en extérieur, la directrice aimerait avoir une solution de secours pour pouvoir la proposer sur Discord au besoin.

Dans votre fichier, ajoutez la ligne suivante avec les autres dépendances.

```
const ytdl = require("ytdl-core");
```

Parmi les commandes à traiter, ajoutez les commandes !caraplay, !caraskip et !carastop. Ces dernières appelleront une fonction **DJCarapuce** et qui prendra en paramètre le message.

Dans un premier temps, recopiez cette ligne avant de déclarer vos fonctions de musique.

```
//On créé une "queue" (file d'attente en français) dans laquelle se trouveront l'ID des serveurs qui utilisent cette option et la liste des musiques
const queue = new Map();
```

Ensuite il vous faudra une fonction qui servira pour faire des redirections vers vos fonctions de musique, en fonction de la commande rentrée. Le but étant de regrouper la gestion des commandes de musique.

A - Ajouter dans la queue

Maintenant créez une fonction qui permettra d'ajouter une musique dans la queue et ensuite de jouer de la musique si le bot n'est pas déjà en train d'en jouer !

Pour ce faire, vous allez créer une fonction asynchrone, c'est à dire qui attendra une réponse d'une requête à un serveur avant de pouvoir continuer.

Pour cela, vous pouvez suivre l'exemple suivant :

```
async function execute(message, serverQueue) {
    try {
      const args = message.content.split(" ");
      //On récupère le salon vocal de la personne qui à entrée la commande et on vérifie qu'on a bien le droit de s'y connecter de d'y parler
      const voiceChannel = message.member.voiceChannel;
      if (!voiceChannel) return message.channel.send("Tu dois d\'abord rejoindre un salon vocal !");
      const permissions = voiceChannel.permissionsFor(message.client.user);
      if (!permissions.has("CONNECT") || !permissions.has("SPEAK")) {
            return message.channel.send("J'ai besoins des droits pour rejoindre et pour parler dans le salon vocal !");
      }
      //On récupère les infos du lien qu'on a reçu et si ce n'est pas une URL ou ID valide ça fera une erreur qui sera chopée par le try catch
      const songInfo = await ytdl.getInfo(args[1]);
      const song = {
            title: songInfo.title,
            url: songInfo.video_url,
      };
      //Si il n'y a pas de file queue, on en construit une et on l'assigne
      if (!serverQueue) {
            const queueContruct = {
                textChannel: message.channel,
                voiceChannel: voiceChannel,
                connection: null,
                songs: [],
                volume: 5,
                playing: true
            };
            const connection = await voiceChannel.join();
            if (!connection) {
                queue.delete(`${message.guild.id}`);
            }
            queueContruct.connection = connection;
            queueContruct.songs.push(song);
```

```
            queue.set(message.guild.id, queueContruct);
            message.channel.send(`[${queueContruct.songs.length}] :
${queueContruct.songs[0].title} a été ajouté à la liste !`);
            serverQueue = queue.get(message.guild.id);
            //On lance ensuite la fonction qui va essayer de jouer
de la musique avec la tête de liste
            play(message.guild, serverQueue.songs[0]);
        } else {
            serverQueue.songs.push(song);
            message.channel.send(`[${serverQueue.songs.length}] :
${song.title} a été ajouté à la liste !`);
        }
    } catch (exception) {
      return message.channel.send({ embed: { color: 16711680,
description: `__**ERREUR**__\nLa commande n\'a pas fonctionnée
<:surprised_carapuce:568777407046221824>\n\n__L\'erreur suivante
s\'est produite :__\n*${exception}*`}});
    }
}
```

B - La fonction play

Il vous reste maintenant qu'à définir vos fonctions pour jouer, passer et arrêter les musiques !

Dans le cas où vous les prenez les dans l'ordre et commencez par la fonction `play()`, la première chose à faire c'est de regarder s'il y a quelque chose dans le paramètre **song** qu'on vous envoie. Si ce n'est pas le cas, ça signifie que la liste de musiques est vide. Dans ce cas, il faudra déconnecter le bot du salon vocal et détruire la queue pour ce serveur.

Pour cela ajouter ce bout de code dans votre fonction :

```
const serverQueue = queue.get(guild.id);

if (!song) {
   serverQueue.voiceChannel.leave();
   queue.delete(guild.id);
   return;
}
```

L'étape suivante consiste à appeler une fonction de la bibliothèque *ytdl-core* pour jouer la musique puis lorsque la musique est terminée, l'enlever de la liste et essayer de jouer la musique suivante.

Pour cela on récupère les différents évènements comme ci-dessous :

```
const dispatcher =
serverQueue.connection.playStream(ytdl(song.url))
   .on("finish", () => {
      serverQueue.songs.shift();
      play(guild, serverQueue.songs[0]);
    })
    .on("error", error => {
      console.error(error);
    });
dispatcher.setVolumeLogarithmic(serverQueue.volume / 5);
```

C - La fonction skip

Le but de cette fonction est de pouvoir passer une musique et donc de passer à la suivante s'il en existe une.

Dans un premier temps, il vous faudra vérifier si l'utilisateur qui veut passer une musique est bien dans un salon vocal et s'il y a bien une musique à passer au moins.

Ensuite il vous faudra enlever la première musique de la liste de musiques du serveur en question et appeler à nouveau la fonction **play()** en lui donnant les arguments qu'il lui faut.

D - La fonction stop

Le but de cette fonction est de pouvoir faire arrêter le bot de jouer de la musique. Pour commencer on vérifie à nouveau que l'utilisateur qui veut arrêter la musique est bien dans un salon vocal. Il suffira ensuite de vider la queue pour ce serveur et de faire s'arrêter la musique qui est en train d'être jouée.

Ce qui donnerait :

```
function stop(message, serverQueue) {
    if (!message.member.voiceChannel)
      return message.channel.send("Tu dois d\'abord rejoindre un
salon vocal !");
    serverQueue.songs = [];
    serverQueue.connection.dispatcher.end();
    serverQueue.voiceChannel.leave();
}
```

N'oubliez pas d'appeler la fonction **DJCarapuce** lorsque qu'un utilisateur enverra !caraplay.

Félicitations, vous avez atteint le niveau 10 !

Vous avez sauvé de nombreuses soirées endiablées !

X - Aller plus loin

Félicitations pour être arrivé jusque-là ! La directrice du camping de Pokéville et son carapuce disent être très fier de vous !

Mais le petit carapuce sait que vous pouvez aller encore plus loin et qu'il ne tient qu'à vous de découvrir les autres possibilités !

N'hésitez surtout pas à améliorer ce que vous avez déjà fait jusque-là et à ajouter de nouvelles fonctionnalités. 😉

Indice : Pensez à la sécurité de votre code. Essayez d'utiliser un fichier JSON (fichier de configuration) dans lequel se trouverait le **token** de votre bot. Mais aussi le préfix de vos commandes (afin de pouvoir avoir un code adaptable (ici c'était **!cara**)). Et pourquoi pas aussi votre ID pour que votre bot vous envoie un message lorsqu'il y a une erreur. 😉
Indice 2 : Vous l'aurez surement remarqué mais dans la partie DJCarapuce, lorsque vous créez un objet pour l'ajouter à votre Map, il y a une propriété `playing: true`, vraiment à tout hasard celle-ci ne servirait-elle pas à savoir si on a mis en pause une musique ? 😉

Par ailleurs, avez-vous testé et vérifié tous les cas d'erreurs, comme utiliser une commande en message privé par exemple ?