Poké-Bot

version 3.4.1

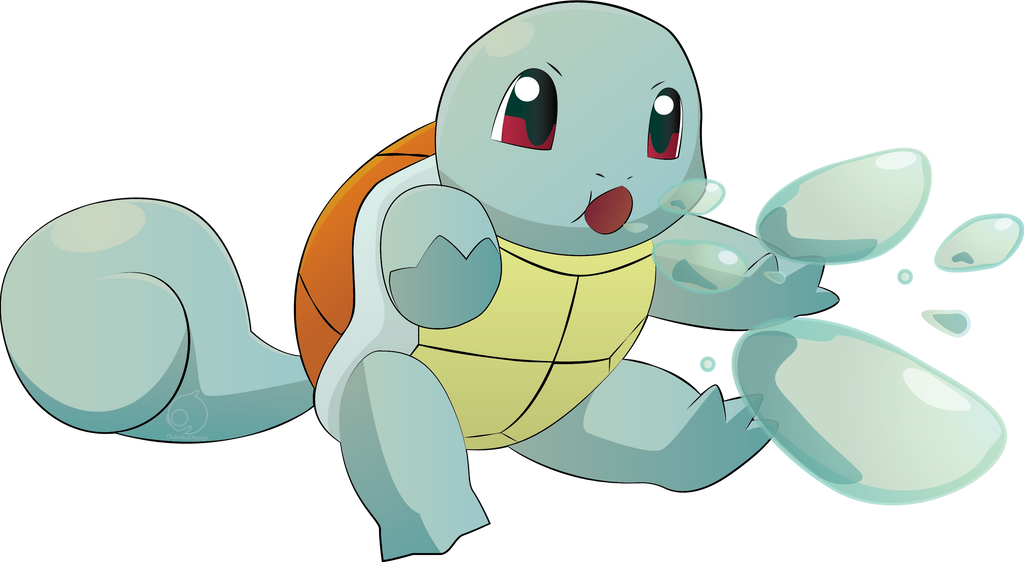




1. Introduction

Pour la première fois depuis des années, l’été atteint des températures particulièrement hautes dans le monde des Pokémons. La directrice du camping de Pokéville aimerait mettre en place un outil formidable : un bot Discord, afin que les habitants de la ville s’amusent durant la canicule. Cette machine révolutionnaire devrait intégrer divers aménagements afin d’organiser différentes activités amusantes, mais la directrice ne s’y connait pas du tout en programmation…

Ayant discuté en avance avec le maire de Pokéville de ce projet, ce dernier souhaite que ce soit vous, éminent ingénieur de la ville, 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* !



Il va rafraîchir l’ambiance !

1. Consignes

* La documentation Javascript pour réaliser le bot est disponible sur [ce site](https://discord.js.org).
* Pour ce projet-là, il vous sera demandé de créer un dépôt avec le nom : « **cc\_bot\_discord\_javascript** ».
* N’oubliez pas de push régulièrement !
* Vous avez tout à fait le droit d’utiliser internet pour trouver des réponses ou pour vous renseigner.
* 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).

⚠ Attention

Dans les balises de code qui vont suivre, les parties en grisprécédées de // sont des commentaires. Ils vous aident à comprendre le code qui suit.

1. Naissance de votre Poké-bot

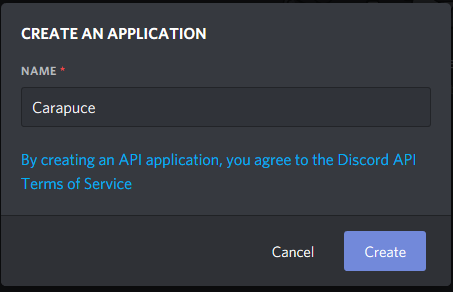
Pour commencer, vous allez créer un bot en allant sur [la page du site officiel de Discord](https://discordapp.com/developers/applications/) dédiée à cet effet. Maintenant identifiez-vous avec votre compte Discord. Si vous n’en avez pas encore, créez-en un.

Graphical user interface, text

Description automatically generated

Le bouton «New Application» se situe en haut à droite

Ensuite, cliquez sur « New Application » et donnez un nom à votre bot : **Carapuce**.



Fenêtre pour créer une application

Afin de lui donner un semblant de vie supplémentaire, donnez-lui une image de profil en cliquant dans l’espace « **App icon** » :

Graphical user interface

Description automatically generated

En voilà une belle bouille !

Grâce aux menus sur la gauche, accédez à l’onglet « **Bot** », puis cliquez sur le bouton « **Add bot** » :

Graphical user interface, text

Description automatically generated

Ajoutons notre Bot.

A screenshot of a computer

Description automatically generated with medium confidenceUne fois cette étape terminée, vous obtiendrez alors quelque chose de similaire à cela :

Votre Bot est prêt à passer à l’action !

1. Viens sur le serv’ !
   1. Et c’est parti !

Maintenant, il va falloir inviter votre bot sur le serveur Discord du camping de Pokéville ! Pour commencer, allez sur l’onglet « **OAuth2** » du menu à gauche. Cochez la case « **Bot** » puis, plus bas, cochez la case Administrateur.

Cela vous permettra de créer un lien d’invitation tout neuf, dédié à votre Carapuce. Ce lien permet à n’importe qui d’inviter le Bot, simplement en le collant dans un navigateur.

A screenshot of a computer screen

Description automatically generated with medium confidence

Le Carapuce est prêt à bondir dans le serveur !

La fenêtre suivante apparaîtra et vous proposera une liste des serveurs sur lesquels vous pouvez inviter le bot ! Ajoutons-le au serveur adéquat. Une seconde fenêtre vous demandera la confirmation des permissions du Carapuce : cliquez sur « **Authorize** ».

Graphical user interface, application

Description automatically generated

Le Carapuce débarque dans le camping !

1. Mise en place du Bot *Carapuce*

Tout d’abord, il vous faudra suivre le [document d’installation de NodeJS](file:///C:/Users/Pierrick%20Mahe/Downloads/Coding%20Club%20-%20Installations%20de%20NodeJS%20et%20ses%20outils.pdf) 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 se trouvent 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.](file:///C:/Users/Pierrick%20Mahe/Downloads/Coding%20Club%20-%20Installation%20de%20FFMPEG.pdf)

1. Premier échange avec le Carapuce

Pour commencer, il va falloir créer un fichier « **Bot.js** ». Ce fichier doit se trouver dans le dossier que vous avez préalablement créé. Dès lors, il vous faudra obligatoirement le code suivant dans votre fichier ! Ce dernier vous permet de vous authentifier auprès de Discord, ainsi que d’intégrer les outils que vous avez installé plus tôt :

// Vous aurez besoin du module discord.js qui appelle l'API de Discord

*const* Discord = require(‘discord.js’);

// Vous appelez une fonction de l'objet créé au-dessus qui permet de

// récupérer un objet utilisateur client qui représente le bot

*const* intents = new Discord.Intents(32767);

*const* bot = new Discord.Client({ intents });

bot.on(‘ready’, () *=>* {

    //Ici vous afficherez dans le terminal que le bot est bien connecté

    console.log(‘Carapuce est dans la place’);

});

// La ligne suivante permet d'indiquer à l'objet Discord qui est votre bot

// afin qu'il puisse se connecter

bot.login("MON\_TOKEN");

⚠ Attention

Votre token correspond à votre « clé d’identification », c’est-à-dire qu’elle va vous servir à connecter votre programme au compte bot. Une personne mal intentionnée pourrait faire n'importe quoi sur les serveurs sur lesquels se trouve votre programme ! Il ne faut donc absolument **JAMAIS PARTAGER** votre token.

Dans le code ci-dessus, « 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, vous le trouverez sous le nom du Bot.

A screenshot of a computer

Description automatically generated  
Votre Token top secret !

Maintenant, essayez d’avoir une première interaction avec Carapuce ! Pour cela, il va regarder les messages envoyés sur les salons textuels du serveur et effectuer une action spécifique en fonction du contenu des messages. Ajoutez à votre script ce bout de code :

// Ici vous regardez quand le bot est en ligne et qu'il voit passer un message (peu importe le serveur)

bot.on("message", *message* *=>* {

    //Vous regardez alors si le contenu du message est exactement "!ping"

    if (*message*.content === "!ping") {

     //Si oui, vous arrivez ici et vous envoyez, dans le même salon, le

       //message "Carapong !"

*message*.channel.send("Carapong !");

    }

});

Maintenant il faut tester pour vous assurer de si ça fonctionne ou non. Pour cela, entrez dans votre terminal 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.

Graphical user interface, application

Description automatically generated

Merci pour cette réponse Carapuce !

Et ça fonctionne, super !Mais ce n’est pas tout ce que vous pouvez faire avec les messages, vous verrez d’autres utilisations dans la partie suivante.



Félicitations, vous avez atteint le niveau 1 !

Vous avez complété les installations pour le bot.

1. Améliorer l’élocution du Carapuce

D’après [la documentation sur les messages](https://discord.js.org/#/docs/main/stable/class/Message), un message est un objet qui a plusieurs propriétés permettant de savoir :

* 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, c’est à dire 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, vous devriez vous intéresser à la méthode « message.author »... Si vous pensez avoir trouvé la bonne réponse, relancez votre bot et si vous obtenez ceci, vous aurez tout fait comme il faut ! Sinon, regardez bien la documentation. La partie suivante apporte une réponse à cette question… à vous de décider si vous souhaitez la voir ou pas !

Graphical user interface, text, application

Description automatically generated

Il commence à devenir bavard !

La directrice du camping aimerait que lorsque quelqu’un salue Carapuce, celui-ci ajoute une réaction au message de salutation. Pour commencer simplement, il vous faudra récupérer l’emoji smiley (😄). Merci aux développeurs de Discord, il existe 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 devriez-vous 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 vous permet de réagir aux messages grâce à un emoji sous forme de phrase, exemple : ’’😄’’, ou grâce à l’ID d’un emoji personnalisé. Pour l’obtenir, procédez de la même manière que pour obtenir l’emoji smiley. Si votre émoji se nomme « :joyeux\_carapuce: », il vous suffira d’ajouter \ devant pour obtenir l’ID de l’emoji.

Text

Description automatically generated  
Aperçu du résultat

Celui-ci sera en deux parties, le nom de l’emoji et un nombre qui est son ID, ici seul l’ID vous intéresse. Ce dernier identifiera l’emoji, même hors du server sur lequel il existe !

Si vous ne savez pas comment ajouter des emojis personnalisés sur votre serveur, cela vous est expliqué dans la partie Poké’mojis. Chose importante à noter, un bot ne peut pas utiliser des emojis personnalisés d’un serveur s’il n’est pas dessus !

bot.on("message", *message* *=>* {

    //Vous regardez alors si le contenu du message est exactement "!ping"

    if (*message*.content === "!carabonjour") {

        //Voilà notre message de réponse !

*message*.reply("Carabonjour à toi!");

        //Carapuce peut réagir avec un émoji de Discord…

*message*.react("😄");

        //Ou avec un émoji personnalisé !

*message*.react("<:56623217846642442:>");

    }

});

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.

A screenshot of a video game

Description automatically generated with medium confidence

*Il réagit désormais au quart de tour* *!*

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, cherchez du côté de « **message.content** » !

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

//Si le tout premier mot du message est « !pin », épingler le message !

if (message.content.startsWith("!pin")) {

    message.pin();

}

Ce qui devrait vous donner comme rendu ceci :

Text

Description automatically generated  
Carapuce sait maintenant épingler les messages !

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.

1. Une mémoire de Carapuce

Le Carapuce de la directrice du camping adore vous regarder coder et voir le projet prendre forme petit à petit ! La directrice étant souvent occupée, celle-ci 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*](https://discordjs.guide/popular-topics/embeds.html#embed-preview) 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 !

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 vous renverrez un message dans le salon textuel, mais le message envoyé sera de type *embed*.

Pour ça, vous pouvez procéder de la manière suivante :

 if (*message*.content === "!carahelp") {

        //on envoie un message de type embed dans le channel d'où provient le

        //message

*const* newEmbed = new Discord.MessageEmbed()

            .setColor('#3498db')

            .setDescription("\_\_\*\*Les différentes commandes\*\*\_\_")

            .setFields(

                {

                    name: "!carahelp",

                    value:"Pour afficher cette aide."

                },

                {

                    name: "!ping",

                    value:"Pong!"

                },

            )

*message*.channel.send({embeds: [newEmbed]});

    }

});

Le résultat sera alors le suivant :

Graphical user interface, text

Description automatically generated  
En voilà une belle mise en page !

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.

Les Embeds ont plein de fonctionnalités et d’options que vous pouvez ajouter. Celles-ci sont listées sur la page d’aide plus haut : n’hésitez pas à en ajouter pour rendre votre Embed plus joli !.



Félicitations, vous avez atteint le niveau 5 !

Vous savez maintenant utiliser des embeds.

1. Poké’mojis

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

Pour commencer, il vous faut des images. Petit conseil, utilisez des images en PNG pour avoir le fond transparent, ou des images « Style émote », comme celles de Twitch par exemple. Sinon, prenez des images de *Carapuce*.

Maintenant, allez dans les paramètres du serveur.

Graphical user interface, application

Description automatically generated  
Les paramètres sont ici

Puis cliquez sur le menu emoji puis Ajouter un emoji.

Text

Description automatically generated

Aperçu de l’onglet Emoji

Il ne vous reste plus qu’à choisir l’image qui servira d’emoji. Puis recommencez pour chaque emoji que vous souhaitez 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 cliquer sur le nom de l’émoji dans le menu, et le 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 :

Graphical user interface, application

Description automatically generated  
Carapuce vous renvoie la liste de ses emojis

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 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**.

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 JavaScript, 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 :

Graphical user interface, application

Description automatically generated

Aperçu des emojis personnalisés

La directrice souhaiterait que le Carapuce puisse réagir avec un emoji personnalisé quand le message contient un mot particulier, comme « carapuce » par exemple. Et surtout, elle 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 ».

⚠ 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.

1. Le Cara-DJ est dans la place !

Une des activités phares des campings ce sont les karaokés ! Dans la crainte de ne pouvoir faire cette activité en extérieur à cause de la chaleur, la directrice aimerait avoir une solution de secours pour pouvoir la proposer sur Discord au besoin.

A cet effet, vous aurez besoin d’un **package**.

Parmi les commandes à traiter, vous devrez créer les commandes « !caraplay », « !caraskip » et « !carastop ». Dans un premier temps, cherchez un package adapté à vos besoins. Petit conseil : effectuez votre recherche en Anglais ! Vous voudrez utiliser un package compatible avec la dernière version de Discord, soit la **v13**.

Si vous avez trouvé [ce package](https://www.npmjs.com/package/discord-music-player), très bien ! Vous avez bien compris l’idée. Vous pouvez ajouter le package avec les lignes suivantes, tout en haut de votre code :

*const* { Player } = require("discord-music-player");

*const* { RepeatMode } = require('discord-music-player');

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. Vous devrez utiliser une fonction dite « **asynchrone** », c’est-à-dire que le programme n’aura pas à attendre le feu vert de la fonction avant de continuer. Votre fonction asynchrone se traduit ainsi :

client.on("messageCreate", async(*message*) *=>* {

});

1. Et c’est parti pour des playlists endiablées !

Maintenant, il vous faut créer une fonction qui permettra d’ajouter une musique dans la queue et ensuite de jouer la musique si le bot n’est pas déjà en train d’en jouer ! N’hésitez pas à jeter un coup d’œil au code exemple de la page du **package**.

Votre fonction devrait ressembler à ceci :

if(command === '!play') {

*let* queue = client.player.createQueue(message.guild.id);

        await queue.join(message.member.voice.channel);

*let* song = await queue.play(args.join(' ')).catch(*\_* *=>* {

            if(!guildQueue)

                queue.stop();

        });

    }

Vous remarquerez que ce code est plutôt compliqué. Il mélange les outils de Discord, de la bibliothèque **discord-music-player**, et les outils de JavaScript. N’hésitez pas à prendre un peu de temps afin de bien comprendre ce segment : vous serrez amené à revoir ces concepts par la suite.

Mais ce morceau de code n’est pas complet ! Vous avez peut-être remarqué que, cette fois, vous n’utilisrez pas « **message.content** ». En effet, les commandes du DJ Carapuce se font en deux parties : la commande, comme « !play », et le nom d’une musique par exemple. Il vous faut donc un moyen flexible de traiter les messages.

Graphical user interface, text

Description automatically generated

Une commande !play des plus classiques.

JavaScript offre plusieurs moyens de traiter un texte tel que celui que vous récupérez. Notre but est de [séparer](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String/split) la commande du reste du message, sachant que cette dernière sera toujours le premier mot reçu. La commande sera stockée dans une variable appelée « **command** », comme au-dessus.

Il vous faudra aussi créer une « queue », ou « file d’attente » en français, qui servira à empiler les demandes de musique, qui seront ensuite jouées à la suite dans l’ordre. Cette ligne devrait faire l’affaire :

*let* guildQueue = client.player.getQueue(message.guild.id);

1. Vas-y DJ, fait péter le son !

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

Le package qui vous a tant donné va cette fois encore faire des miracles. Ce dernier inclut des fonctions existantes pour toutes nos envies ! Commencez avec des fonctionnalités simples, comme celles listées ci-dessus. N’oubliez pas, [le package est votre meilleur ami !](https://www.npmjs.com/package/discord-music-player)

A picture containing clipart

Description automatically generated

Félicitations, vous avez atteint le niveau 7 !

Vous avez sauvé de nombreuses soirées endiablées !

1. Le Cara-quizz

Tout comme le Karaoké, les Quizz du camping se font en plein air. Cependant, chacun chante bien mieux au frais qu’en pleine canicule ! Il faut donc que les Quizz soient faisables avec Discord… Un jeu d’enfant pour vous à présent !

Le résultat doit être similaire au suivant :

Graphical user interface, text, application, chat or text message

Description automatically generated

Aperçu du quiz de Carapuce

Text

Description automatically generated

Attention, c’est compliqué !

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

Pour commencer, écrivez un code qui permettre à Carapuce de répondre aux messages « !caraquizz ». Vous devez commencer à connaitre ça !

Ensuite, vous allez créer un fichier « **questions.json** ». Il contiendra les questions et les réponses qui seront utilisées durant le quizz. Une fois créé, il faudra appeler votre fichier comme vous avez appelé un package ; ce dernier sera interprété comme un [tableau](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array). Votre fichier devrait ressembler à ceci :

Text

Description automatically generated

J’espère que vous connaissez vos classiques !

Vous avez tout pour bien démarrer. A présent, vous devez indiquer au Carapuce de choisir une question au hasard parmi celles proposées. JavaScript a tout ce qu’il vous faut : la fonction **Math.random()** permet de tirer un chiffre qui déterminera la question posée.

//Ici, notre fichier de questions s’appelle simplement ‘questions’

*var* rnd = Math.floor(Math.random() \* questions.length);

//’questions’ est un tableau, ce qui signifie qu’on peut y accéder avec des chiffres correspondant aux différentes cases du tableau.

*message*.channel.send({ content: questions[rnd].question })

//On va récupérer les réponses des participants au quizz.

*const* answerCollector = *message*.channel.createMessageCollector({filter, time: 10000});

💡 Information : Si vous ne comprenez pas une partie du code, n’hésitez pas à chercher sur internet !

Juste au-dessus, vous avez créé un [Collector](https://discordjs.guide/popular-topics/collectors.html#message-collectors). Regardez bien la page associée, et essayez de l’utiliser. Si vous ne trouvez pas, vous pouvez utiliser ça :

answerCollector.on('collect', *answer* *=>* {

answerCollector.stop();

Il est temps de finaliser le quizz ! Vous avez récupéré une réponse, mais comment savoir si c’est la bonne ? Vous allez créer une boucle comme suit :

for (*var* x in questions[rnd].answer) {}

Cette boucle permet de parcourir toutes les réponses. Maintenant, il faut les comparer avec la réponse donnée par l’utilisateur ! Vous aurez besoin de la fonction **.include()** de JavaScript, ainsi que **.toLowerCase()** afin de rendre tout ça plus facile à utiliser (« carapuce » sera la même chose que « CarApuCe »). C’est une condition un peu complexe ; la réponse est en dessous.

if(*answer*.content.toLowerCase().includes(questions[rnd].answer[x].toLowerCase(

)))

{

     return *message*.channel.send("Your answer is right!");

}

return *message*.channel.send("Your answer is wrong :(");

Voilà une condition plutôt longue !

Notre Caraquizz est prêt à l’emploi ! Une fois les questions ajoutées, Carapuce est prêt à les poser ! Ajoutez la commande correspondante dans la fonction que vous avez créé plus tôt, et testez vos connaissances à des questions que vous avez-vous-même créé !

A picture containing clipart

Description automatically generated

Félicitations, vous avez atteint le niveau 8 !

C’est l’heure de tester vos connaissances !

1. 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.

* Pensez à la sécurité de votre code. Essayez d’utiliser un fichier JSON dans lequel se trouverait le token de votre bot. Mais aussi le préfix de vos commandes, afin de pouvoir avoir un code adaptable. Et pourquoi pas aussi votre ID pour que votre bot vous envoie un message lorsqu’il y a une erreur.
* Avez-vous bien compris tout ce que vous avez écrit ? Essayez de reprendre votre code et de vous imaginer la démarche que suit le bot durant l’exécution. Les fonctions que vous avez utilisées peuvent-elles être améliorées ? Cherchez les pages de manuels sur internet.

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