Rapport sur la conception du bot et de l'interface pour le projet 'Bot de modération Discord'

Dans le cadre de notre projet web, nous cherchons à mettre en place un bot sur l'application de chat en ligne **discord**, le but étant de mettre en place un bot pouvant réaliser les tâches suivantes :

> - Gérer la modération en permettant au bot de créer ou supprimer des modérateurs sur le serveur Discord
> - Gérer les sanctions en permettant au bot de punir des utilisateurs en recevant une commande d'un modérateur
> - Détecter les comportements illicites tels que le spamming et le punir en conséquence sans passer par le biais d'un modérateur. En prime, nous mettrons en place une interface web permettant de gérer le bot sur un site à part, ce site web disposera des fonctionnalités suivantes :
> - Permettre la diffusion publique du bot
> - Chaque utilisateur ayant le bot installé sur son serveur doit avoir accès à un panel d’administration. Ce panel d’administration doit permettre d’activer/désactiver ou/et de configurer certaines fonctionnalités du bot. Notre mission doit donc nous amener à comprendre le fonctionnement d'un serveur node js, et établir un lien entre notre application et un élément extérieur.

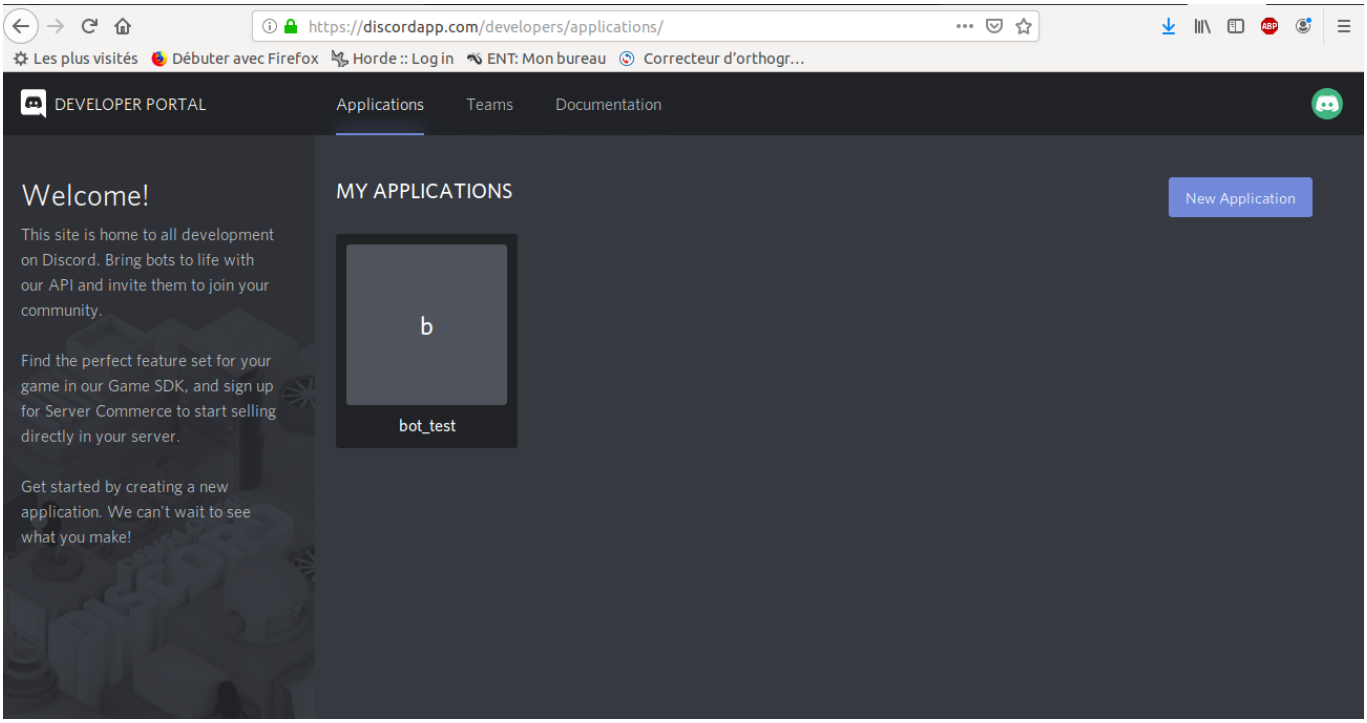
Au travers de ce rapport nous verrons la manière dont nous avons réalisé ces tâches, en commencant par l'élaboration d'un serveur node js avec un webSocket, puis la réalisation du bot avec Discord.js et enfin la réalisation de notre panel de gestion du bot.

Pour toutes questions en ce qui la base de données, vous pouvez vous référer à notre précédent rapport `rapport_conception_db.md`.

Création d'un bot Discord

La première partie de notre travail de création d'un bot Discord est évidemment de créer le bot. Pour ce faire nous nous renderons sur le portail Developers de discord à l'adresse :

'https://discordapp.com/developers/applications/'



On crée une application qui deviendra notre futur bot. Discord nous fournit ce portail afin de simplifier la création d'application sur leur plateforme.

On se rend ensuite dans l'onglet **"Bot"** afin de confirmer que notre application est un utilisateurs robot, une fois que l'on confirme la nature de notre application nous serons données par Discord un `Token` qui sera utilisé par notre application node js afin d'appeler le bot sur un serveur.

Mise en place du WebSocket

Afin de comprendre le principe derrière la discussion entre node js et Discord, nous avons commencé par établir un webSocket qui nous permettrait d'établir une discussion avec le serveur discord, mette en ligne notre bot et mettre en place une fonctionnalité qui retournerait un message **'Pong'** en cas de réception d'un message **'Ping'**.

L'intérêt de cette partie est double, comprendre la discussion entre node js et Discord par le biais de node js, et tester la fonctionnalité du bot vu précédemment.

Le code originel du ping pong est disponible dans le fichier ***/pingpong/index.js***.

Fonctionnement du WebSocket :

Le but du WebSocket est de mettre en place une liaison entre l'api Discord et notre fichier node js. Pour ce faire nous commencons par déclarer un webSocket `client`, ce webSocket nouis permet de faire le lien avec discord par le biais de la `gateway` discord.

Si la connexion est réussie, le webSocket va vérifier en permanence ce qu'il reçoit de la gateway et, en fonction de son résultat, appelera une fonction `message()` qui vérifiera le contenu du message et en fonction de ce dernier pourra effectuer diverse action.

Si le message envoyé au WebSocket est un message écrit, il peut être comparé au sein de la fonction `onMessage()` afin de vérifier s'il s'agit d'une commande.

On extrait avec `substring` pour vérifier la présence du caractère du commande '`!`'. En cas de présence du caractère, on passera par la commande ***RegExp***.

RegExp

Au vu de son importante capital dans notre projet, regExp méritait sa propre catégorie.

RegExp est un constructeur JavaScript permettant de reconnaitre une chaine de caractère afin d'en extraire des informations nécessaire au bon déroulement de la commande :

> *Exemple :*
>
> message.match(new RegExp('^!ping[ ]*$', 'i')))
>
> `message.match()` : Check si le message est correcte par rapport à la fonction en paramètre
>
> `RegExp()` : Check si le message correspond au string en paramètre
>
> `'^!ping[ ]*$'` : si le message correspond à ce pattern alors le regExp retourne vrai
>
> `'i'` : on indique que le message doit être converti en minuscule, sans majuscule

Avec ce que nous avons pu tirer de notre travail sur le webSocket et les différents composants nécessaire, nous avons enfin pu nous lancer sur la réalisation de notre bot d'administration Discord.

Conception du bot avec Discord.js

> Afin de faciliter la réalisation de notre bot, nous avons choisi d'utiliser la bibliothèque **discord.js**.

Grâce à l'outil discord.js, nous pouvons simplifier la mise en place du bot, à l'aide des différentes fonction déjà présente dans l'extension. Cependant la simplification ouvre la prote à l'étendu du problème de la réalisation de l'application et afin de la simplifier, nous avons découpé l'application en plusieurs dossiers, chacun devant traités un aspect spécifique du bot :

index.js

Le fichier en charge de traiter la connexion du bot au serveur, ainsi que la récupération des messages addressé au bot avant de les envoyer vers un index de traitement. Cela inclut notamment la détection de si le message est destiné au bot et vient d'un autre bot (dans ce cas il est ignoré) ou d'un utilisateur humain. Dans ce cas le message est envoyé dans la commande `check_and_run()` de src/command.index.js afin de vérifier si le message vient d'un propriétaire légitime ou non.

Le fichier index.js est disponible dans /src/

command/index.js

Le fichier index.js dans ./src/command/ à pour but de traiter les commandes envoyées par un utilisateur. Il commence d'abord par vérifier si l'utilisateur est légitime dans sa demande en vérifiant s'il dispose des droits nécessaires.

S'il dispose des droits nécessaires la commande est passée au travers d'un moulinage de **Regex** afin de comprendre de quelle commande il s'agit. Si la commande est trouvée, elle est immédiatement exécutée par

l'appel d'une fonction `callfunc()` qui traitera la fonction reçue.

Si le match() ne retourne rien, alors il ne s'agit pas d'une fonction n'est pas traiter.

Les commandes globales

Nous nous sommes très vite rendu compte que pour effectuer les commandes de manière optimisée dans leurs traitements, il était nécessaire de créer des fichiers personalisés pour chaque commande globale. Les commandes globales traitent un problème de manière précise et unique en fonction de la commande global appelé par le **regex**.

> Voici la liste des commandes globales et leurs fichiers de traitement :

- Bannir un utilisateur :
  `ban.js`
- Exclure un utilisateur :
  `kick.js`
- Rendre sourd un utilisateur :
  `deaf.js`
- Rendre muet un utilisateur :
  `mute.js`
- Avertir un utilisateur :
  `warn.js`
- Déclarer un channel comme channel de logs pour le bot :
  `setlogchannel.js`
- Annuler une sanction par son id :
  `cancel.js`
- Ajouter un role de moderation à un utilisateur :
  `rankup.js`
- Retirer un role de moderation à un utilisateur :
  `delrank.js`
- Ajouter un rôle (le créer) :
  `addrole.js`
- Supprimer un rôle :
  `delrole.js`
- Ajouter une commande à un rôle :
  `role_add.js`
- Retirer une commande à un rôle :
  `role_del.js`
- Récupérer les sanctions appliquées à un utilisateur :
  `getto.js`
- Récupérer les sanctions appliquées par un modérateur :
  `getfrom.js`
- Verouiller un ou des channels :
  `lock.js`
- Déverouiller un ou des channels :
  `delock.js`

- Supprimer les messages d'un channels (message d'un joueur et/ou depuis x sec) : `delmsg.js`
- Obtenir la liste des commandes sur le serveur : `help.js`

> *Description de `lock.js` :*
>
> lock.js fût le premier nécessitant la création d'un rôle afin de le faire fonctionner.
>
> En effet, afin de verrouiller les channels, il est nécessaire de bloquer les utilisateurs en leur retirant les droits de lecture, écriture sur le channel, cependant cela ne pouvait pas être fait sur le rôle @everyone car cela aurait posé beaucoup plus de travail sur lock.js pour le mettre en place.
>
> Au lieu de cela, nous avons créer un rôle lock, de priorité très forte et on le donne à tout le monde. Ainsi, ceux ayant ce rôle se voit incapable d'écrire/lire sur le channel. Bien sûr, l'administrateur et les modérateurs autorisés peuvent continuer à écrire/lire dessus.

> *Description de `cancel.js` :*
>
> cancel.js est un fichier nous permettant d'annuler une sanction d'après son id dans la base de données. Après avoir exécuter une commande dans la bdd pour vérifier l'existence de la sanction, cette dernière est effacé de la base et en fonction de son type on applique différent traitement permettant d'annuler les punitions appliquées.
>
> Noter que cancel.js efface aussi la commande des logs.

> *Description de `setlogchannel.js` :*
>
> Une commande particulière qui permet de transformer un channel en un channel de logs, cela signifie que le bot emploiera ce channel pour décrire toute les actions qu'il effectue.
>
> Pour mettre en place ce channel, la commande accède à la base de données et à l'intérieur du serveur lui indique quel channel (donc l'id du channel) auquel il doit envoyer envoyer les logs du bot.
>
> Si aucun channel de log n'est en place, alors le bot n'envoi aucun log.

> *Description de `help.js` :*
>
> Une commande permettant de récupérer de la base de données l'ensemble des commandes présente sur le serveur.

Chacune de ces commandes est retrouvable au sein du dossier ***/command/*** et dispose de sa propre manière de traiter l'information reçu en paramètre. Elle retourne toujours un message en cas d'erreur.

Panel

Le panel est la troisième et dernière partie de notre projet. Nous devons mettre en place une interface afin de pouvoir administrer le bot depuis une page web.

Cette interface comporte une page d'accueil par lequel nous nous connections à discord ainsi que d'un lien permettant d'appeler le bot sur votre serveur Discord.

Le panel est retrouvable dans le dossier /src/web et est composer de