PROJET BDD+WEB
Création d'un bot de modération Discord.

Ce document est composé de cinq parties :
0 – Introduction
1 – Présentation du service Discord
2 – Le cahier des charges
3 – Le plan de travail
4 – Annexes

Ce projet est composé de deux grandes parties évaluées séparément :
1 – Réflexion et Création d'une base de donnée.
2 – Création d'un bot de modération en NodeJS et de son site web/panel d'administration.

Règles du jeu

La réalisation de ce projet se fait en duo.
Pour mener le projet à bien, il va falloir vous organiser en identifiant les différentes tâches à effectuer et suivre leur réalisation. Vous pouvez utiliser des outils de gestion de projet.

Attention, la partie sur la base de données (PARTIE 3 - I) doit être réalisée par les deux membres du duo dans un premier temps.
Seulement lorsque celle-ci est validée, vous pouvez passer aux parties suivantes.
Le non respect de cette règle est sanctionné par un nul.
La partie PARTIE 3 - II et la partie PARTIE 3 - III peuvent-être réalisées simultanément.

Énoncé

Mise en contexte : Plusieurs de vos amis ont créé différents serveurs Discord : pour partager leur passion d'un jeu vidéo, pour organiser leurs factions, pour parler & partager différentes informations concernant les cours de l'ENSSAT etc…

Certains de ces serveurs marchent vraiment bien, et possèdent un grand nombre d'utilisateur ! Malheureusement avec un grand pouvoir, viennent de grandes responsabilités ! Certains des membres insultent, ou/et partagent des photos indélicates. Il arrive aussi que des hordes d'utilisateurs humains ou robotiques s'amusent à RAID votre serveur Discord tard le soir :'(. Bonjour la pluie de notification ! Cela suffit !

Vous vous êtes décidé à créer un service permettant la gestion & la modération d'un serveur Discord.

Au fait ! Qu'est-ce que Discord ?

« Discord est basé sur un principe de serveur, que n'importe quel utilisateur peut utiliser, ajuster et rejoindre. Sur ces serveurs, les administrateurs peuvent créer des salons vocaux ou textuels et définir des permissions pour chaque utilisateur sous forme de rôle. Discord fournit du texte riche avec une syntaxe Markdown. » - https://fr.wikipedia.org/wiki/Discord_(logiciel)

PARTIE 1 :
Présentation d'éléments concernant le service Discord.

I - Discord
Source : https://support.Discordapp.com

Discord peut être décrit comme un ensemble de salons et d'utilisateurs.

1) Les utilisateurs

Un utilisateur du service Discord posséde un ID unique (numérique), et un pseudonyme constitué d'un pseudo et d'un numéro d'identification (pseudo#num).
Toutefois même si ce pseudo est unique il n'est pas constant dans le temps. Les utilisateurs peuvent changer leur pseudo, et les membres ayant payé un abonnement à Discord nitro peuvent modifier leur numéro d'identification. Donc se servir de celui-ci pour identifier un utilisateur n'est pas une bonne idée.

Un utilisateur du service Discord est invité à rejoindre un serveur Discord (guild dans la documentation du service) . S'il accepte l'invitation, il devient membre du dit serveur Discord. Un membre d'un serveur Discord est donc un couple (Utilisateur Discord + Serveur Discord). Il faut faire la distinction entre un utilisateur du service et un membre d'un serveur. Un membre possède des droits particuliers sur un serveur, un utilisateur non.

Se référer à l'annexe A pour un exemple.

2) Les rôles

Un utilisateur a un ou plusieurs rôles associés sur un serveur Discord. Par défaut, l'ensemble des utilisateurs est associé au rôle @everyone. Un rôle peut être décrit comme un tuple : (Position dans la hiérarchie, Nom, Couleur, paramètres, Permissions).

Les rôles fonctionnent selon une hiérarchie linéaire, les membres d'un serveur héritent des propriétés (permissions, couleur, ect…) du rôle le plus haut placé dans la liste des rôles qui leurs sont assignés.

paramètres d'un rôle :
- afficher séparément les membres ayant ce rôle des autres membres en ligne.
- permettre à tout le monde de mentionner ce rôle.

Permissions :

+ Générales :
- Administrateur
- Voir les logs du serveur
- Gérer le serveur
- Gérer les rôles
- Gérer les salons
- Expulser des membres
- Bannir des membres

- Créer une invitation
- Changer de pseudo
- Gérer les pseudos
- Gérer les émojis
- Gérer les webhooks
- Lire les salons textuels & voire les salons vocaux

+ Salons textuels (cf. Les salons/Permission spéficiques aux salons textuels)
+ Salons vocaux (cf. Les salons/Permission spéficiques aux salons vocaux)

3) Les salons

Un salon peut être décrit comme un tuple : Nom, Type, Catégorie, paramètres, Permissions associées.

Type : Les salons vocaux ou les salons textuels. Les paramètres & permissions associés à un salon dépendent de son type.

Pour les salons textuels :	Pour les salons vocaux:
+ paramètres : - Description du salon - Slowmode - Salon NSFW	+ paramètres : - Taux d'échantillonage - Nombre d'utilisateur maximum
+ Permissions : - créer une invitation - gérer le salon - gérer les permissions - gérer les webhooks.	+ Permissions : - créer une invitation, - gérer le salon, - gérer les permissions, - gérer les webhooks.
+Permission spéficiques aux salons textuels : - lire les messages, - envoyer des messages, - envoyer des messages TTS, - gérer les messages, - intégrer des liens, - attacher des fichiers, - voir les anciens messages, - mentionner @everyone, - utiliser des émojis externes, - ajouter des réactions.	+Permission spéficiques aux salons vocaux: - voir le salon - se connecter - parler - rendre des membres muets - rendre des membre sourds - déplacer les membres - utiliser la détection de voix - priority speaker

Les permissions liées aux salons sont prioritaires sur les permissions liées aux rôles.
Par exemple, le rôle @everyone a la permission de lire l'ensemble des salons par défaut.
Il est possible d'enlever la permission « Lire les messages » au rôle @everyone (salon devient invisible et inaccessible) et de donner celle-ci uniquement à certains rôles.

Note sur les catégories : Les salons sont rangés dans des catégories. Il est possible de définir des permissions pour les salons textuels et vocaux pour chaque catégorie. Ainsi chaque salon d'une catégorie peut-être défini comme ayant les même permissions que celle-ci (salon synchronisé) ou

il peut être configuré pour ne pas prendre en compte les permissions de celle-ci (salon non-synchronisé).

II – Sanction

Par défaut, Discord offre deux sanctions possibles : bannir un utilisateur du serveur ou bien expulser un utilisateur du serveur.

Le bannissement n'a pas de durée, ainsi celui-ci est définitif tant que personnene la lève. Tant que l'utilisateur est banni celui-ci ne pourra pas rejoindre votre serveur.

Expulser un utilisateur revient à le forcer à quitter votre serveur. C'est à dire que celui-ci n'est plus membre de votre serveur, il devra retrouver un lien d'invitation valide pour rejoindre votre serveur.
La dureté d'une telle sanction dépend de la disponibilité de lien d'invitation vers votre serveur et du nombre d'utilisateur de celui-ci. Les serveurs Discord ayant une capacité maximum d'utilisateurs, expulser un utilisateur revient à libérer une place vers celui-ci, pour un utilisateur peut-être moins indélicat.

Le contournement des sanctions Discord est une chose plutôt aisée, le service n'offrant pas de ban IP ou machine, la création d'un nouveau compte Discord permet de rejoindre à nouveau celui-ci. Pour tempérer cela, Discord offre la possibilité d'ouvrir son serveur uniquement aux personnes ayant validés leur compte via leur numéro mobile. L'utilisation d'un numéro mobile permet de limiter le nombre de compte Discord qu'un utilisateur peut créer.

Pour infliger d'autres types de sanction aux utilisateurs, on utilise couramment un rôle adapté.
Si l'on souhaite créer une sanction « muet », on pourrait faire comme suit :
- créer un rôle « muet » qui n'a aucune des permissions « envoyer des messages », « envoyer des messages TTS » et « se connecter ».
- le rôle « muet » doit être supérieur dans la hiérarchie aux différents grades des joueurs.
Ainsi, un utilisateur ayant le rôle « muet » ne pourra plus écrire/parler dans un salon textuel/vocal.

PARTIE 2 :
Cahier des charges.

I-Introduction

Le but est de réaliser un bot de modération Discord et un site web/panel d'administration.
Attention, le bot de modération doit pouvoir être utilisé sur différents serveurs Discord.

Le bot de modération doit offrir différents services :
1) Gestion des modérateurs
2) Gestion des sanctions
3) Détection automatique de certains types de comportement
4) Gestion des logs

Le site web lié au bot doit offrir différents services :
1) Permettre la diffusion publique du bot
2) 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.

II – Le bot de modération

La modération doit se faire via le bot, c'est à dire qu'un modérateur doit écrire une commande qui va être interprétée par le bot. Le bot réagira à la commande en considérant celle-ci et le grade du modérateur. Il est important de noter que la manière dont réagit un bot est particulière au serveur sur lequel il est, puisque son comportement doit être paramétrable pour chaque serveur.

1) Comment modéliser un bot de modération ?

Dans un premier temps, nous pourrions définir les actions élémentaires que nous aimerions que notre bot soit capable de faire. Le bot doit avoir la permission administrateur. Grâce au bot de modération, les utilisateurs ayant un rôle de modération n'ont pas à avoir de permission Discord particulière. Le bot reconnaît les membres ayant un rôle de modération, et exécute les commandes que ceux-ci écrivent dans un salon.

1.a) Particules de modération

Les particules correspondent à l'ensemble des types de sanctions pouvant être appliquées par le bot.

o AVERTIR : Un message est envoyé au joueur pour le prévenir qu'il a fait une bêtise.
o MUET : Le joueur ne peut plus envoyer de message et/ou est rendu muet sur un salon vocal.
o SOURD : Le joueur ne peut plus consulter un salon et/ou est rendu sourd sur un salon vocal.
o EXCLURE: Le joueur est exclu du serveur Discord.
o BAN : Le joueur est banni du serveur Discord.

Pour que celles-ci soient réellement effectives il faut définir à quel utilisateur elles s'appliquent et indiquer la motivation de la sanction. Les sanctions peuvent aussi être tempérées en ne les rendant pas définitives, c'est à dire en spécifiant une durée d'application.

1.b) Spécifieurs

Les spécifieurs permettent d'associer à une particule différentes informations telles que l'utilisateur subissant la sanction, la durée et la raison de celle-ci.

+Sanctions définitives

o D[particule](user, reason) Sanction définitive de l'utilisateur *user* sur l'ensemble du Discord, pour la raison *reason*.

o D[particule](user, reason,channels) Sanction définitive de l'utilisateur *user* sur l'ensemble des salons dans la liste *channels*, pour la raison *reason*.

+Sanctions temporaires

o T[particule](user, duration, reason) Sanction temporaire d'une durée *duration* de l'utilisateur *user* sur l'ensemble du Discord, pour la raison *reason*.

o T[particule](user, duration, reason,channels) Sanction temporaire d'une durée *duration* de l'utilisateur *user* sur l'ensemble des salons dans la liste *channels*, pour la raison *reason*.

+ Informations concernants les différents paramètres :

- *User* correspond à l'identifiant numérique d'un utilisateur Discord (unique & constant).
- *Duration* correspond à la durée de la sanction. En pratique, nous indiquerons le timestamp de la fin de la sanction.
- *Reason* correspond à une chaîne de caractères libre, par laquel le modérateur indique la raison de la sanction.
- *Channels* liste des salons sur lesquels la sanction est appliquées.

1.c) Résolveurs

pseudo(@pseudo) → ID unique Discordapp.com
durée(1d) → durée en s.
canaux(*Audio) → les canaux vocaux
canaux(*Texte) → les canaux textuels
canaux(#name) → le canal du nom name
canaux(*cat) → les canaux de la catégorie cat
canaux(?perso) → les canaux d'une liste personnalisée (perso)

1.d) Les commandes de modération.

Nous souhaitons que l'administrateur d'un serveur Discord utilisant notre bot de modération puisse paramètrer les commandes disponibles pour la modération et l'effet de celles-ci. L'effort de modélisation précédent nous permet d'exprimer une solution simple à ce problème.

CMD(ModérateurRP,!ban @user duration reason)

De plus il est possible d'ajouter des restrictions sur les arguments de la commande.
CMD(ModérateurRP,!ban @user duration<=durée(1d) reason)

Exemples : Dans le cadre d'un serveur Discord RP (Role Play), nous pouvons avoir des salons liés au RP et des salons de discussion générale. L'administrateur de ce serveur Discord souhaite créer deux grades : Modérateur et Maitre du Jeu.
Le maitre du jeu est là pour faire respecter les règles du RP. Il sanctionne les joueurs s'obstinant à ne pas vouloir suivre celles-ci. Les sanctions des maitres du jeu s'appliquent donc seulement sur les

salons RP. Le modérateur sanctionne le mauvais comportement en général. Et donc, les sanctions des modérateurs s'appliquent sur l'ensemble des salons du Discord.

Les deux grades doivent avoir accès aux deux commandes suivantes, toutefois leur effet doit différé selon leur rôle :
!ban @user duration reason .
!mute @user duration reason .

Dans le langage de formalisation vu précédement les besoins de l'administrateur se transcrivent comme suit :
CMD(ModérateurRP,!ban @user duration reason) :- T[BAN](user(@user),durée(duration), reason, canaux(?RP)).
CMD(ModérateurRP,!mute @user duration reason) :- T[MUTE](user(@user),durée(duration), reason, canaux(?RP)).
CMD(Modérateur,!ban @user duration reason) :- T[BAN](user(@user),durée(duration), reason).
CMD(Modérateur,!mute @user duration reason) :- T[MUTE](user(@user),durée(duration), reason).

1.e) Communiquer la sanction

Lorsqu'un joueur est sanctionné, le bot doit lui envoyer un message privé sur discord contenant différentes information concernant sa sanction.
Donc, lors de la création d'une commande, il doit être possible de configurer différents messages : le message envoyé à la personne sanctionné, et le message de confirmation au modérateur.
Pour cela l'administrateur doit écrire un template pour les différents message envoyés.

Par exemple, l'administrateur doit pouvoir configurer les messages pour qu'il s'affiche comme suit :
- Si le « ModérateurRP » Cédric#0001 effectue la commande «!ban @user 1d test », l'utilisateur @user devrait recevoir le message suivant de la part du bot :

Suite à la décision du **ModérateurRP Cédric#0001**, vous avez été banni de façon __temporaire__ **des salons RP du serveur Kawaii Army** pour la raison : **test**.
Votre ban expire le **16 décembre 2020.**

- Si le « Modérateur » Cédric#0002 effectue la commande «!ban @user 1d test», l'utilisateur @user devrait recevoir le message suivant de la part du bot :

Suite à la décision du **Modérateur Cédric#0002**, vous avez été banni de façon __temporaire__ **du serveur Kawaii Army** pour la raison : **test**.
Votre ban expire le **16 décembre 2020.**

De plus on souhaite envoyer un message à une personne écrivant une commande.
Ce message permet de confirmer l'exécution d'une commande ou indique une cause d'erreur doit être configurable par l'administrateur, comme l'est un message envoyé à une personne sanctionnée.
De plus on souhaite pouvoir afficher un message permettant de communiquer la validation ou la non validation d'une commande dans le tchat où la commande a été executée, ou bien en envoyant un message privé.
Les messages d'erreurs doivent informer qu'une commande n'est pas disponible si l'utilisateur n'a pas le droit de l'utiliser, ou qu'elle a déjà été appliquée, ou qu'elle est erronée.

2) Gestion de la modération :

Comme nous l'avons vu précédement, nous aimerions créer différents rôles de modérateur auxquels nous aimerions affilier des permissions pour des commandes spécifiques. Par exemple, dans la section 1.e selon le rôle de l'utilisateur la commande « !mute » provoque une sanction différente.

- Pour ajouter l'un des grades de modération à un joueur /rankup @user rôle ou retirer un rôle /derank @user rôle.
- Un modérateur doit accèder à l'ensemble des sanctions qu'il a infligées via une commande.

3) Détection automatique de comportement abusif

Les comportements problématiques que l'on souhaite détecter :
- Utilisation de mot vulgaire
- Spam : correspond à la publication répétitive d'un même message dans un laps de temps cours.
- Flood : correspond à un message contenant une répétition abusive d'une même lettre ou mot.

Il existe différentes manières de détecter ces comportements, de l'écriture de règles permettant de définir celles-ci à l'utilisation de réseau de neurone.
Dans le cadre de ce projet, il vous est recommandé d'utiliser des règles.

La détection de mot vulgaire peut se faire via l'utilisation d'une liste de mots vulgaires. L'utilisation de règles regex permettent de dériver différentes variantes de la liste. Le but étant de détecter à partir d'un mot 'gros mot' l'ensemble des variantes 'gro mot' 'gro mo' 'grau mau' 'grot mt' *ect.*…

La détection de flood peut se faire via des règles regex sur les messages. Par exemple, en vérifiant si le message n'est pas la répétition d'un même pattern.

La détection de spam peut se faire via l'analyse des messages d'un utilisateur, en vérifiant si les différents messages sont identiques et/ou le temps entre l'envoi de deux messages.
Par exemple, un utilisateur qui envoie en quelques secondes une dizaine de messages comportant 40/50 caractères est très certainement en train de spammer.

4) Gestion des sanctions :

Nous aimerions pouvoir vérifier les sanctions passées d'un joueur, ou celles infligées par un modérateur spécifique. Donc il nous faut les commandes suivantes :
- Accéder aux sanctions d'un joueur spécifique
- Accéder aux sanctions délivrées par un modérateur
- Il doit être possible de lever une sanction. Par exemple, une commande générique qui permet de lever n'importe quelle sanction.

5) Autres fonctionnalités du bot :

+Gestion des logs :
1) L'ensemble des commandes effectuées doivent être loggé.
2) Si un utilisateur supprime ou modifie un message cela doit être loggé.
3) Les logs sont des messages envoyés par le bot dans un salon spécifique défini par l'administrateur.

+Gestion des salons :
1) Il doit être possible de verrouiller/déverrouiller un salon, ou un groupe de salons via une commande.
2) Il doit être possible de supprimer l'ensemble des messages publiés dans un salon pour une durée déterminée et/ou des joueurs spécifiques.

III – Le panel d'administration/Site web

Le site web a deux fonctions principales. La première est de mettre à disposition le lien d'invitation permettant aux joueurs d'inviter le bot sur leur serveur. La seconde est de permettre à ceux-ci d'administrer le bot pour leur serveur.

Le site web est composé :
- d'une page d'accueil comportant le lien d'invitation & un accès au panel d'administration
- d'un panel d'administration permettant aux joueurs d'administrer le bot pour leur serveur.

NB : La connection au panel d'administration pourrait se faire en utilisant son compte Discord .

1) La page d'accueil
La page d'accueil doit contenir deux éléments principaux :
- un lien pour inviter le bot sur son serveur Discord
- un moyen de se connecter au panel d'administration en s'identifiant.

2) Le panel d'administration
Le panel d'administration doit permettre de configurer le bot, c'est à dire

- de créer/supprimer différentes commandes accessibles via le bot.
- de créer/supprimer différent grade de modération.
- d'associer au grade de modération les différentes commandes auxquelles il a accès.
- d'activer/désactiver les logs et indiquer dans quel salon ceux-ci doivent être inscrit. Il doit être possible de différencier différents type de log, on peut souhaiter afficher dans deux canaux séparés les logs de modération et les logs concernant les modifications des messages.
- activer la détection automatique de certaines infractions & configurer la sanction associée.

PARTIE 3 :
Plan de travail

I- Je commanderais une base de donnée, s'il-vous-plait !

A – Proposer un schéma Association-Entités + dictionnaire de données.
B – Traduire le schéma en schéma logique, expliciter les règles utilisées.
C – Écrire un script permettant de remplir la base
D – Écrire les différentes requêtes utiles pour la suite du projet. Chaque requête doit être écrite dans un fichier SQL séparé.

Aide :
> Un membre d'un serveur Discord correspond à un utilisateur Discord ayant rejoint le dit serveur Discord.

> Pour chaque serveur Discord, il existe différents grades de modération avec un accès à différentes fonctionnalités (commandes, salons).

> Pour chaque serveur Discord, l'administrateur peut créer&configurer différentes commandes pour différents rôles de modération.

Requêtes minimales devant être traitées durant cette partie :

01 - Récupérer l'ensemble des serveurs Discord ayant invité le bot.
02 - Récupérer l'ensemble des grades liés à la modération d'un serveur Discord.
03 - Mettre à jour le rôle d'un membre à grade lié à la modération.
04 - Récupérer l'ensemble des modérateurs liés à un serveur Discord.
05 - Appliquer une sanction à un utilisateur sur un serveur Discord.
06 - Récupérer l'ensemble des sanctions appliquées sur un serveur Discord.
07 - Récupérer la liste des joueurs dont les sanctions sont toujours actives sur un serveur Discord.
08 - Récupérer la liste des joueurs étant connectés sur au moins deux serveurs Discords.
09 - Récupérer les sanctions liées à un joueur, et leur nombre.
10 - Récupérer la liste de sanctions infligées à un joueur par un modérateur.
11 - Récupérer la liste de sanctions infligées par un modérateur et leur nombre.
13 - Afficher la liste de sanctions infligées sur un serveur
15 - Récupérer les joueurs ayant des sanctions sur plusieurs serveurs.
16 - Récupérer la liste de joueur ayant des sanctions sur différents serveurs partageant une temporalité commune.
17 - Récupérer la liste de joueur ayant des sanctions similaires sur différents serveur

II- Let’s build some bot !
La partie 2 correspond à la création du bot Discord, tel que décrite dans la partie « Travail demandé ».

L’api du service Discord se trouve à l’adresse suivante :
https://Discordapp.com/developers/applications/ .
Se référer à l’annexe B pour plus d’information concernant l’API REST.

Le bot devra être réalisé en NodeJS. Les frameworks ne sont pas acceptés par défaut, seul les frameworks suivants sont acceptés : express, cors, ejs, reactjs, node-cron.
Si vous souhaitez utiliser un autre framework, vous devez demander l’autorisation auparavant.

III- Le panel d’administration/Site Web

La partie 3 correspond à la création du panel d’administration/Site web, tel que décrite dans la partie « Travail demandé ».

Le côté client devra être écrit en NodeJS avec les restrictions précédentes sur les frameworks. Toutefois pour la partie client vous êtes libres de vos choix.

IV - BONUS : Live long & prosper.

Notre bot permet d’avoir une modération efficace du serveur, et apporte quelques outils de gestion. Dans cette partie nous allons ajouter quelques fonctionnalitées au bot.

+ Accueil des nouveaux membres :
Le but étant d’accueillir les nouveaux membres via un message du bot Discord.
Si l’administrateur le souhaite, les membres n’ont pas accès aux salons discord hormis le salon règlement ; tant qu’il ne l’ont pas accepté.

+ Commandes personnalisées :
L’administrateur peut créer différentes commandes de modération via l’interface développée précédement. Celles-ci sont toutefois limitées à la modération . Il serait intéressant d’offrir la possibilité de rajouter des commandes de « gestion » .

Il serait intéressant d’avoir un couple de commande /tag name /settag name , permettant de créer des messages automatiques. La commande /settag alert ‘Ceci est une alerte !’ permet d’instancier le tag ‘alert’ , ainsi lorsqu’une personne ayant la permission de faire la commande /tag ; utilise /tag alert ; le bot écrira dans le salon « Ceci est une alerte ! »

Comme les commandes précédentes la commande /tag alert doit pouvoir être réservée à certains rôle et/ou dans certains salons.

+ Gestions des invitations :
Lorsqu’une invitation est générée elle est associée à celui qui en a fait la demande. Lorsqu’un nouvel utilisateur arrive on identifie via quel invitation il est arrivé. Cela permet de faire un classement entre les membres qui invitent le plus de personnes sur le serveur Discord. Mais cela permet aussi de savoir qui a invité qui. Cela doit aussi nous permettre de désactiver une invitation.

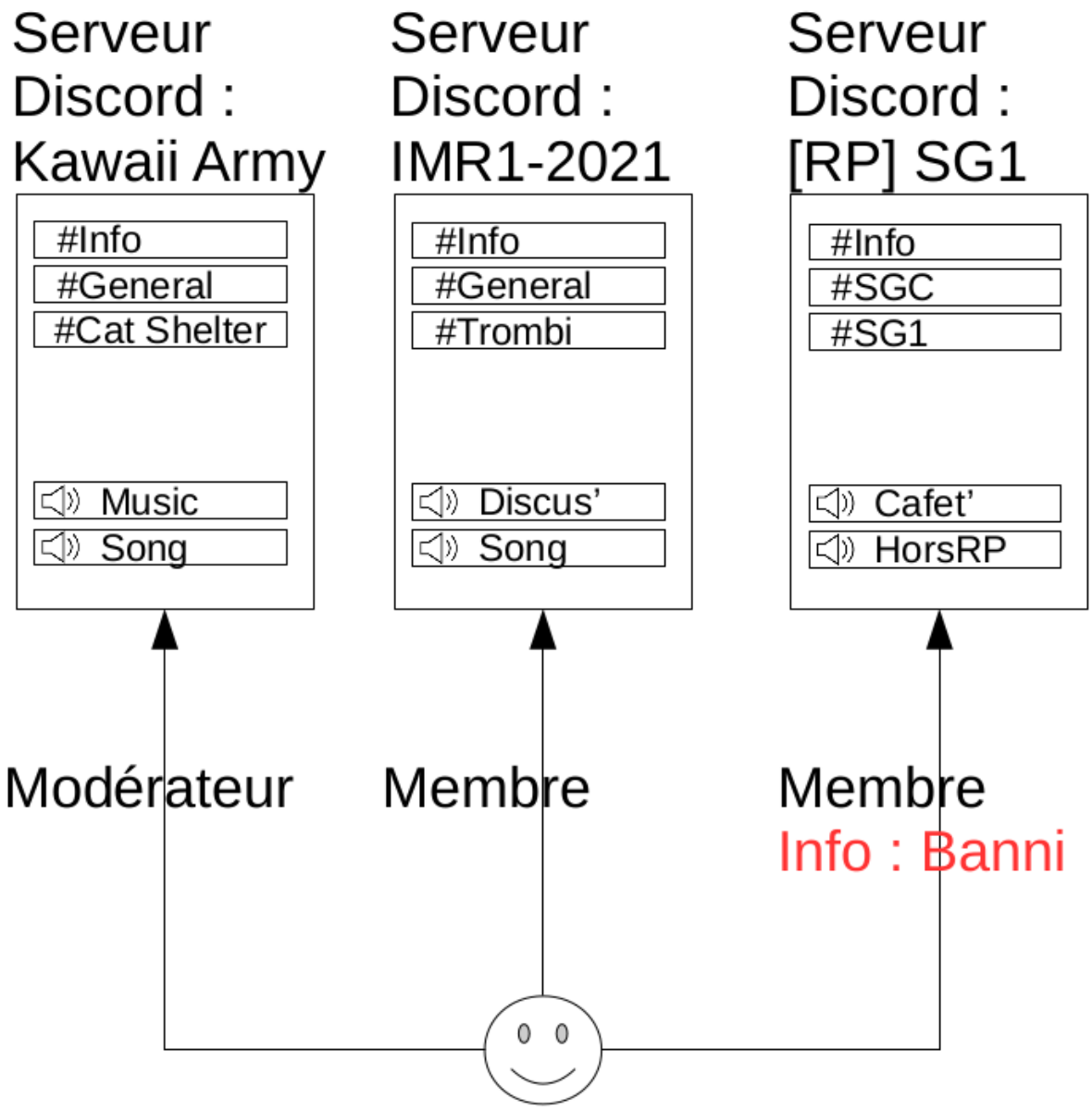
+ Gestion de rôles personnalisés :
Nous aimerions créer différents rôles liés à certaines statistiques propres à un utilisateur. Par exemple, selon le nombre de messages que celui-ci a envoyés, ou bien le nombre de personne que celui-ci a acceptées ou bien même son nombre de sanction .

+ Visualisation & Statistiques
Affichage de statistiques concernant le serveur Discord : nous aimerions avoir différents graphes concernant l'activité du Discord dans le panel d'administration.

Nous aimerions avoir une carte de l'ensemble des permissions accordées sur le serveur via le bot & les permissions natives, accessible via le panel d'aministration.

ANNEXES

Annexe A : Organisation du service Discordapp.com



Annexe B : Relation entre URI et REST

URI	GET	POST	PUT	PATCH	DELETE
Ressource collection, telle que `http://api.example.com/collection/`	*Récupère* les URI des ressources membres de la ressource collection dans le corps de la réponse.	*Crée* une ressource membre dans la ressource collection en utilisant les instructions du corps de la requête. L'URI de la ressource membre créée est *attribué automatiquement* et retourné dans le champ d'en-tête *Location* de la réponse.	*Remplace* toutes les représentations des ressources membres de la ressource collection par la représentation dans le corps de la requête, ou *crée* la ressource collection si elle n'existe pas.	*Met à jour* toutes les représentations des ressources membres de la ressource collection en utilisant les instructions du corps de la requête, ou *crée éventuellement* la ressource collection si elle n'existe pas.	*Supprime* toutes les représentations des ressources membres de la ressource collection.
Ressource membre, telle que `http://api.example.com/collection/item3`	*Récupère* une représentation de la ressource membre dans le corps de la requête.	*Crée* une ressource membre dans la ressource membre en utilisant les instructions du corps de la requête. L'URI de la ressource membre créée est *attribué automatiquement* et retourné dans le champ d'en-tête *Location* de la réponse.	*Remplace* toutes les représentations de la ressource membre, ou *crée* la ressource membre si elle n'existe pas, par la représentation dans le corps de la requête.	*Met à jour* toutes les représentations de la ressource membre, ou *crée éventuellement* la ressource membre si elle n'existe pas, en utilisant les instructions du corps de la requête.	*Supprime* toutes les représentations de la ressource membre.

La méthode GET est sûre, c'est-à-dire que l'appliquer sur une ressource ne résulte pas en un changement d'état de la ressource (sémantique de lecture seule). Les méthodes GET, PUT et DELETE sont idempotentes, c'est-à-dire que les appliquer plusieurs fois sur une ressource résulte en le même changement d'état de la ressource que les appliquer une seule fois, bien que la réponse puisse différer. Les méthodes GET et POST sont stockables en cache, c'est-à-dire que le stockage des réponses à ces requêtes pour une future réutilisation est autorisé.

Contrairement aux services web orientés SOAP, il n'y a pas de norme officielle pour les API REST, parce que REST est une architecture alors que SOAP est un protocole. REST n'est pas une norme en soi, mais les implémentations qui suivent cette architecture utilisent des normes comme HTTP, URI, JSON et XML13.

Extrait: https://fr.wikipedia.org/wiki/Representational_state_transfer