LUSTGARTEN Léo

Projet Tutoré
- Bot Discord -

Introduction

- **Description**

Le but de ce projet est de créer un bot discord permettant aux utilisateurs d'un serveur de simplifier les comptes d'une ou plusieurs cagnottes, et d'avoir facilement accès aux informations sur celle-ci. L'utilisation se fait à l'aide de commande à entrer directement sur un serveur Discord, dans un canal d'écriture.

Les utilisateurs peuvent :
- Créer une cagnotte
- Participer à une cagnotte
- Entrer leurs dépenses
- Calculer les remboursement à faire entre les participant
- Fermer une cagnotte

Ainsi que plusieurs commandes d'affichage qui seront détaillées par le suite.

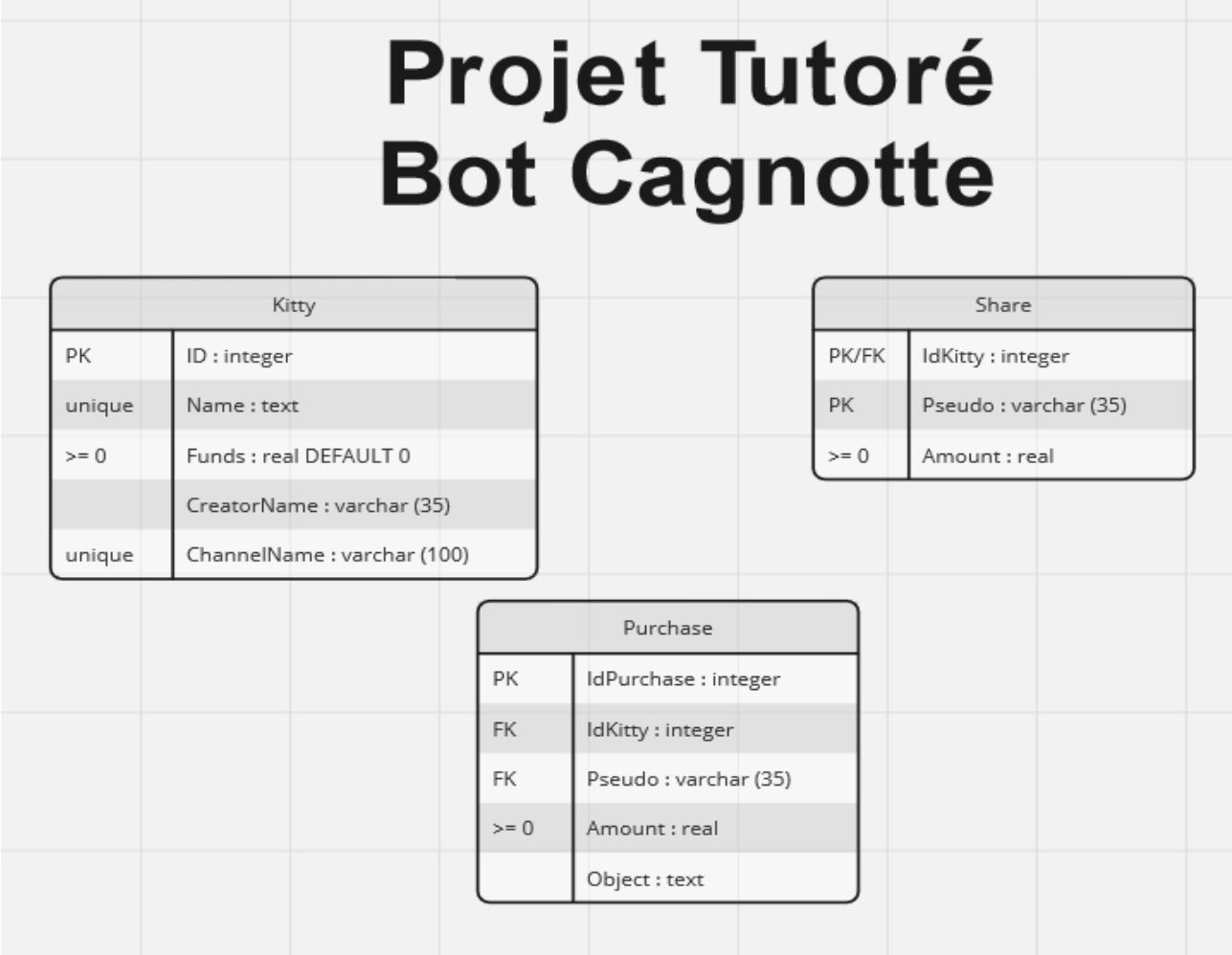
- **Techno utilisées**

Les langages couramment utilisés pour la programmation de Bot Discord sont Javascipt et Python. J'ai donc choisi python, car son utilisation m'est un peu plus familière.

Afin de stocker les informations liées aux cagnottes il me fallait aussi choisir une base de données. Pour celle ci j'ai suivis les conseils de mon tuteur, et ai utilisé SQLite, n'ayant pas besoin de stocker de grandes quantités de données.

J'utilise la bibliothèque discord.py pour les fonctionnalités liée à discord, ainsi que aiosqlite pour interagir avec ma base de donnée SQLite.
J'utilise aussi la bibliothèque dotenv et le module os de la bibliothèque standard pour récupérer des informations sensibles dans un fichier .env

Base de donnée



- **Précisions sur les types**

Les varchar on une valeur qu'il n'est pas possible de dépasser. Discord n'autorise pas de pseudo avec plus de 35 caractères ni de nom de channel de plus de 100 caractères.

Ayant appris avec PostgreSQL je souhaitait utiliser le type « numeric(5,2) » pour les montants d'argents, mais ce type n'existe pas sur SQLite. Le type permettant de stocker des nombres à virgules est donc « real ».

Il n'existe pas non plus de type « serial » en SQLite, cependant un integer en clef primaire s'auto incrémente naturellement. Cependant si une ligne est supprimée, la valeur libérée peut être réutilisé. Pour que ce ne sois pas le cas il faut rajouter « AUTOINCREMENT » à la fin de la ligne, mais n'en ayant pas spécialement l'utilité, je me suis contenté de l'auto incrémentation de base.

- **Précisions sur les contraintes**

La contrainte unique dans la table Kitty porte sur les deux champs en même temps. Les noms de cagnotte sont donc unique par canal.

Présentation et Utilisation

Pour utiliser le bot il faut l'inviter sur un serveur, puis renseigner l'id de notre serveur dans le fichier .env (cela permet de synchronisé les commandes à notre serveur)
Ensuite il suffit d'entrer les commandes dans un canal écrit du serveur

- ## Créer une cagnotte

/createkitty [kitty_name]
Permet de créer une cagnotte en lui donnant un nom. Le bot renvoie un message pour confirmer la création de la cagnotte, ou pour indiquer qu'elle n'a pas pu être créer (la cagnotte [kitty_name] existe déjà)

- ## Participer à une cagnotte

/participate [kitty_name] [amount]
Permet de participer à un cagnotte en indiquant le nom de la cagnotte et le montant que l'on souhaite. Il est possible de diminuer sa participation en entrant une valeur négative, sauf si l'argent a déjà été dépensée. Le bot renvoie un message pour préciser si la participation a été valider ou invalider.

- ## Acheter quelque chose pour une cagnotte

/purchase [kitty_name] [object] [amount]
Permet d'acheter quelque chose pour une cagnotte, en précisant le nom de la cagnotte, l'objet de la dépense ainsi que son montant. Encore une fois le bot renvoie un message de réponse.

- ## Calculer les transactions de remboursement

/calculate [kitty_name]
Permet de calculer et d'afficher les transactions minimum à faire pour que les participant d'un cagnotte donnée se remboursent.

- ## Fermer une cagnotte

/kittyclose [kitty_name]
Permet de fermer une cagnotte donnée. Les données ne sont pas supprimer mais la cagnotte devient inaccessible.

- ## Afficher les informations d'une cagnotte

/showkitty [kitty_name]
Affiche les informations d'une cagnotte donnée. Son créateur, le total des participations, le total des dépenses, et les fonds restants

- ## Afficher la liste des participants et participations d'une cagnotte

/showshares [kitty_name]
Affiche chaque participation à une cagnotte donnée (participant et montant).

- ## Afficher la liste des achats d'une cagnotte

/showpurchase [kitty_name]
Affiche les achats d'une cagnotte donnée (acheteur, objet et montant)

- ## Afficher les cagnottes auxquelles on participe

/kittyme
Affiche toutes les cagnottes créer par l'utilisateur, ainsi que toutes les cagnottes auxquelles il a participé (l'utilisateur est le seul à voir la réponse du bot)

Organisation et Avancement

Au lancement de ce projet mon tuteur m'a conseillé de découper mon projet en plusieurs phases.

- ## Phase 1: Un utilisateur gère la cagnotte seul

Un seul utilisateur gère la cagnotte seul :
- Ajouter le bot sur un serveur
- Créer une cagnotte
- Ajouter des participations (créateur) //gardé par la suite pour l'ajout de participation extérieur)
Exemple : /participe [nom] [montant]
- Ajouter des dépenses (créateur)
Exemple : /achat [nom] [montant]
- Afficher les informations d'une cagnotte
Exemple : /afficher [nomCagnotte]
Le total des participations est de : [montant]
Vous avez dépensé : [montant]
Il reste donc : [montant]
- Afficher les noms des cagnottes en cours
- Calculer les remboursement à faire (pour qu'il y est le moins de transaction possible)
- Fermer la cagnotte

Voici le contenue de la première phase. Chaque objectifs a été rempli, il y a cependant quelques différences qui on été opéré pendant mon avancement. Les commandes ne sont pas uniquement réserver au créateur de la cagnotte (hormis la fermeture). Les commandes ou un nom devait être entré en argument n'ont finalement pas besoin de ce nom car j'ai découvert qu'il était possible de tout simplement récupérer le pseudo de l'utilisateur ayant entré la commande. Par ce même principe le nom du canal ou la commande est entrée est récupéré, je n'avais pas pensé à cet fonctionnalité au début mais il se trouve qu'elle est très importante pour gérer l'accès aux cagnottes. En effet si le but d'une cagnotte est de faire une surprise à un autre membre du serveur, il serait embêtant qu'il puisse y avoir accès. Une cagnotte est donc restreinte au canal ou elle est créée et apparaît inexistante depuis les autres canal (sauf pour la commande kittyme car sa réponse n'est pas visible par tous)

- ## Phase 2 : Participant et Achat précis

- Voir à quelles cagnottes on participe (éphémère)
Exemple : /moi
Anniversaire Gus
Surprise Leonore
(synthèse de combien on a donné comme participation (l'argent qu'on doit potentiellement) / combien on a payé (et donc l'argent qu'on nous doit))
- Chacun peu mettre sois même sa participation
Exemple : /participe [montant] // Récupération auto du pseudo de l'utilisateur
- Ajouter un objet à la liste des objectifs de cagnotte (/add nomObjet, prix, […])
- Acheter un objet de la liste
Exemple : /achat [nomObjet]
- Ajouter un moyen de transaction préféré pour les participants *(Paypal, virement, liquide…)*

La phase 2 n’a malheureusement pas encore énormément avancer. J’ai ajouter en plus de celle présenter ici d’autre fonction d’affichage décrite plus haut, afin que l’utilisateur puisse avoir une meilleure visualisation d’un cagnotte en cours. Les listes des objectifs et moyen de transaction préféré n’ont pas encore été ajoutés. Leurs ajout demande de modifier la base donnée et j’ai malheureusement été pris par le temps. Cependant je souhaite continuer le projet (au moins jusqu’à la fin de la phase 2), et j’ai pensé à une autre fonctionnalité que je souhaiterai ajouter en phase 2 : La possibilité d’entrer des « amis proches » dans la base de donnée, et ainsi modifier la fonction de calcul des transactions pour que les utilisateur puissent rembourser en priorité un amis proche si c’est possible. Les transactions ne seront alors pas forcement minimum mais cela permet de simplifier les échanges au seins d’un groupe de personnes en fonction de leurs affinités. Par exemple si je souhaite faire une cagnotte pour faire une surprise à mon frère et qu’il y a d’autres membres de ma famille ainsi que des amis à lui, il est généralement plus simple que les membres de la famille se remboursent entre eux si possible.

Aussi et en priorité, il serait intéressant d’ajouter une commande « kittyhelp » pour afficher toutes les fonctionnalités et comment s’en servir

- ## Phase 3 : Droits et modifs

3 niveaux : Créateur, Admin, Utilisateur (modifier en conséquences les droits des actions précédentes)
- Ajouter/retirer un admin (créateur)
- Retirer/Modifier une participation (Admin)
- Retirer/Modifier un paiement (Admin)
- Renommer la cagnotte (Admin)

- ## Phase 4 : Améliorations

- Envoyer en privé les infos des dettes
/ exemple un utilisateur recevra un message qui récapitule ce qu’il doit aux autre utilisateurs (ou ce qu’on lui doit), et quels sont les moyens de paiement que préfères ces autres utilisateurs */*
- Ajouter des dates d’échéances et des rappels
- Ajouter des auto complétions (les noms des cagnottes, les pseudos)

N’ayant pas terminé la phase 2 je ne me suis pas du tout penché sur les phases 3 et 4. Mais comme prévu par le découpage du projet en phases, leur contenu est tout à fait optionnel (il s’agit plus de bonus au cas ou le projet aurait été trop simple)

Conclusion

Les débuts du projet on été assez difficile, je mettait les pieds dans un milieu inconnu, avant même de commencer à coder j'ai du me renseigner sur comment coder un bot, comment l'inviter sur un serveur etc..
Ensuite il a fallu lire beaucoup de doc et regarder quelques tutos afin de comprendre réellement ce que je faisais. J'ai fait plusieurs test et codé plusieurs fonctions sans rapport pour me faire la main avant d'attaquer réellement le code du projet. De plus il existe différentes manière de faire, j'ai commencer par créer des « commandes préfixe » par exemple avant d'apprendre que cette manière de faire était plutôt déprécier désormais. J'ai d'ailleurs laissé à disposition de l'ancien code dans un dossier « old » ainsi que le fichier testCog dans les plugins (qui peut être rajouter au main en seulement quelques lignes de code) au cas ou j'aurais besoin de faire d'autres testes.

J'ai aussi rencontrer quelques problèmes avec ma base de donnée, plusieurs fois j'ai essayer de faire des choses que je savais faire sous PostgreSQL, avant d'apprendre que ce n'était pas possible en SQLite.
Les types comme présenté plus haut. Les triggers en SQLite sont très limités, je n'en ai finalement pas fait (j'ai opté pour d'autres solutions). Les suppressions en cascade ne fonctionne pas par défaut et malgré plusieurs essaie avec des solutions trouvé sur internet je n'ai pas réussi (Mon tuteur m'a par la suite conseiller de ne pas supprimer les données, je n'en ai finalement pas eu besoin)

Une fois le projet lancé, le code en lui même n'a pas été trop compliqué (excepté pour le calcul des transactions). La structure d'une commande se répète, la difficulté a plus été dans la réflexion sur certains cas que je n'avais pas imaginé à la base (la séparation par canal par exemple a été trouvé en cours de route), et à toujours faire attention qu'un utilisateur ne puisse pas avoir un comportement inattendu qui ferais planter mon code.

J'ai beaucoup aimé faire ce projet malgré ses difficultés inédites :
- Pas de sujet mûrement réfléchis par les professeurs pour me dicter quoi faire (et malgré mon organisation de départ j'ai fait fasse à plusieurs situations que je n'avais pas imaginé avant d'y faire face)
- Pas de cours, si ce n'est mes bases en python et en SQL, il m'a fallu trouver beaucoup d'information par moi même

Je suis content que l'option projet tutoré m'ait permis de travailler sur un projet qui m'intéresse et me fasse découvrir de nouvelles choses. Il n'est pas impossible que je me mettent à coder d'autres bot Discord pour mon usage personnel à l'avenir grâce aux connaissances acquise pendant ce projet.