Labo 1 - Debt Manager

Informations Générales

- **Date du rendu** : Lundi 16 octobre, 13:15 CEST
- **Groupes** : à réaliser seul ou à deux
- **Plagiat** : en cas de copie manifeste, vous y serez confrontés, vous obtiendrez la note de 1, et l'incident sera reporté au responsable de la filière, avec un risque d'échec critique immédiat au cours. Ne trichez pas. *(Notez que les IAs génératives se trouvent aujourd'hui dans une zone qui est encore juridiquement floue pour ce qui est du plagiat, mais des arguments se valent à en considérer l'utilisation comme tel. Quoiqu'il en soit, nous vous proposons une autre vision sur la question : votre ambition est d'apprendre et d'acquérir des compétences, et votre utilisation éventuelle de cet outil doit refléter ceci. Tout comme Stackoverflow peut être à la fois un outil d'enrichissement et une banque de copy-paste, faites un choix intentionnel et réfléchi, vos propres intérêts en tête, de l'outil que vous ferez de l'IA générative)*

Debt Manager

Nous vous demandons d'implémenter un gestionnaire de dettes entre amis. Ce service sera constitué d'un groupe de personnes, pouvant chacune se connecter à un même serveur à l'aide d'un client en ligne de commande, et lui communiquer les paiements qu'elle aura fait au nom d'un (sous-)ensemble du groupe. Le serveur sera responsable de maintenir à jour les dettes dues entre les différents membres du groupe, et les simplifier afin d'éviter un trop grand nombre de dépendances.

Notez que ce premier labo sera par la suite modifié et enrichi par les suivants. Par conséquent, assurez-vous d'avoir des bases saines, notamment avec une bonne seperation of concerns.

Spécification fonctionnelle

Utilisateur.trice

Chaque utilisateur ou utilisatrice est identifiée par un pseudonyme ne contenant pas d'espace.

Client

Le client prend en argument, à son lancement, les informations suivantes, et dans cet ordre :

- le pseudo identifiant l'utilisateur ou l'utilisatrice au nom de laquelle il effectuera les commandes qui lui seront données,
- l'url complet (port inclu) sur lequel il trouvera le serveur.

Un certain nombre de commandes sont disponibles à travers le client :

- `pay <sum> for <pseudo>[, <pseudo>[, ...]]` permet à l'utilisateur ou l'utilisatrice actuelle d'informer le serveur qu'elle a effectué un paiement de `<sum>` au nom d'un certain ensemble de personnes du groupe, listées par leur pseudo, séparés par des virgules. Le client doit afficher `Success` à l'utilisateur lorsque le serveur a confirmé la réussite de la commande, ou un message d'erreur s'il y en a une.

- `get debts [<pseudo>]` permet de lister les personnes à qui une somme est due, ainsi que cette somme, par la personne identifiée par le pseudo fourni, ou par l'utilisateur ou l'utilisatrice actuelle si aucun pseudo n'est fourni. Le client doit afficher une ligne par dette sous la forme `<pseudo>: <somme>` suivi de la somme totale des dettes, ou un message d'erreur s'il y en a une.
- `get credit [<pseudo>]` permet de lister les personnes qui doivent une somme, ainsi que cette somme, à la personne identifiée par le pseudo fourni, ou à l'utilisateur ou l'utilisatrice actuelle si aucun pseudo n'est fourni. Le client doit afficher une ligne par dette sous la forme `<pseudo>: <somme>` suivi de la somme totale des dettes, ou un message d'erreur s'il y en a une.
- `exit` permet de quitter le programme, et toute potentielle connexion encore ouverte avec le serveur.

Parce que certains tests seront automatisés, assurez-vous de bien suivre le format de ces commandes. Aucun input ne doit être attendu de la part de l'utilisateur outre les commandes ci-dessus.

Exemple d'exécution

```
$ ./client jessie localhost:3333
> pay 32 for jessie, blake
Success
> get debts
ollie: 4.95
blake: 25
29.95
> get credit
parker: 12.45
12.45
> exit
$
```

Serveur

Configuration

Le serveur prend en argument, à son lancement, le chemin d'un fichier de configuration, contenant les informations suivantes, au format JSON :

- Un booléen indiquant si le serveur doit être lancé en mode "debug", dans lequel le serveur pourra afficher sur `stdout`, de manière lisible, autant d'informations que souhaité sur ses actions.
- Le port sur lequel il doit recevoir les nouvelles connections des clients.
- La liste des pseudos existants (un pseudo ne pourra pas, pour l'instant, être ajouté une fois le serveur lancé).
- Pour chaque pseudo, toutes les dettes, décrites par le pseudo de la personne créancière ainsi que la somme due.
  Notez que cette configuration décrit l'état du serveur à son lancement, et n'a pas pour vocation d'être modifié par le serveur. Cela signifie que tout changement (notamment dû à un nouveau paiement) ne sera reflété qu'en mémoire, et sera perdu une fois le serveur clos. Ceci est dans un soucis de simplicité et de facilitation des tests.

Parce qu'elle sera utilisée pour l'automatisation de tests, assurez-vous de bien suivre le format de ce fichier.

Exemple de fichier de configuration

```
{
        "debug": true,
        "port": 3333,
        "users": [
        {
            "username": "blake",
            "debts": [
                {
                    "username": "jessie",
                    "amount": 32.45
                },
                {
                    "username": "ollie",
                    "amount": 12.50
                },
            ]
        },
        {
            "username": "jessie",
            "debts": []
        },
        {
            "username": "ollie",
            "debts": [
                {
                    "username": "blake",
                    "amount": 4
                },
            ]
        },
    ]
}
```

Simplification des dettes

Les dettes forment un graphe dirigé, où chaque noeud représente une personne, et chaque arête une dette due par une personne à une autre. Ce graphe doit être maintenu dans une état simplifié. Nous considérerons ce graphe comme "simplifié" si le nombre total d'arêtes est inférieur au nombre de membres du groupe, et si aucune somme n'est due à une personne qui doit de l'argent. Voir l'appendice pour une preuve par construction de l'existence d'une telle simplification du graphe.

Tests

Nous vous demandons d'écrire des tests unitaires et d'intégration automatisés. Nous vous recommandons de tirer profit des fonctionnalités de Go telles que les interfaces afin de faciliter l'implémentation de tests,

notamment l'utilisation de mocks.
Vos tests devront passer avec le Data Race Detector de Go.

Rendu

Votre repository contiendra notamment les fichiers suivants, que nous pourrons utiliser pour tester votre code.

- Un fichier `test.sh` exécutable permettant de lancer tous les tests, et prenant en argument un booléen indiquant si les tests doivent être lancés avec le flag `-race` pour utiliser le Data Race Detector de Go.
- Un fichier `compile_client.sh` compilant le client et prenant en argument le chemin à donner à l'exécutable.
- Un fichier `compile_server.sh` compilant le serveur et prenant en argument le chemin à donner à l'exécutable.
- Un fichier `README.md` décrivant dans les grandes lignes la structure de votre solution (ses différents packages, goroutines, et leurs dépendances et communications).

Contraintes supplémentaires

Votre code sera évalué pour sa qualité. Nous attendons donc notamment des noms de variable et de fonction clairs, des commentaires pertinents et un README utile.

Toute synchronisation doit être faite à l'aide des goroutines et des channels. Tout autre moyen de synchronisation est interdit.

Pour finir, nous réitérons ici l'importance de vous conformer aux spécifications ci-dessus sur les formats d'input et d'output, puisqu'ils seront utilisés pour automatiser certains tests.

Appendice

> Proof of upper bound on simplified debt graph size
>
> **Claim: There exists a graph such that $E \leq V$ to represent any debt situation within a group of people.**
>
> Let us present an algorithm that finds a simplification of any debt graph, such that it satisfies the inequality in the proof.
>
> In an initial phase, the algorithm computes the total amount of money owed by each vertex, allowing for negative values if that vertex is owed more money than it owes. Let $d(v)$ represent this value, for every $v \in V$. Note that $\sum_{v \in V} d(v) = 0$. This can be done trivially by computing the sum of the amounts of money attached to each in- and out-going edges of every vertex. Note that this partitions vertices into those that owe, those that are owed, and those that are neither -- in other words, no vertex can both owe and be owed.
>
> Consider now the following greedy algorithm that describes new edges, assuming old edges are now obsolete and removed. For each vertex $v \in V$ such that $d(v) > 0$, i.e. v owes money, iterate through each vertex $w \in V$ that is owed money and do the following: letting $d'(w)$ be the sum of

ingoing edges of w, if $d'(w)$ is not $d(w)$, create an edge between v and w with value equal to $d(v, w) = min(d(v), d'(w))$, letting $d'(v)$ be $d(v) - d(v - w)$. To state this in a simple manner, let v give as much money to w until either w is no longer owed any money, or v has given all the money it owes to the group.

Note that, because the sum of $d(v)$ for all $v \in V$ is 0, each vertex that owes money, once iterated over, will have given all its owed money, and consequently each vertex that is owed money will be paid back.

Let us now count the number of edges. To this end, let us show that the creation of an edge always results in the satisfaction of at least one vertex, where by satisfaction we mean that that vertex has paid or been paid everything it owed or was owed. Indeed, there are exactly two situations in which an edge can be created:

- the vertex that owed money was able to give the whole amount of money it owed to the target vertex, and it is now satisfied, or
- the vertex that owed money was not able to give all the money it owed because the target vertex was not owed as much, and the target vertex is satisfied.
  Because a vertex cannot be satisfied more than once, this proves that the number of edges cannot be larger than the number of vertices, which concludes the proof.

As a corollary, note that such a simplified debt graph is bipartite.