Rapport

ON EN A GROS (SUR LE COMPTE)

L2 CUPGE

2019

« Drosophile » - Philippe Truillet

Kregit

Clement Truillet
Valentin Frydrychowski

https://github.com/ctruillet/KreGit/
https://ctruillet.github.io/KreGit/

Table des matières

« Le gras, c'est la vie ! »

- Karadoc

Présentation du projet

Objectif du projet

L'objectif de ce projet est de se familiariser à l'écriture d'un programme à plusieurs avec partage des tâches tant au niveau de la programmation que de la documentation. Il va s'agir de produire un programme (fichiers source et un exécutable testé et opérationnel avec la documentation), dont le sujet est présenté dans la suite du document. Le développement de ce programme se fera obligatoirement sous environnement Linux, en langage C.

Cahier des charges

Réalisation d'un programme en langage C permettant de gérer un système de comptes bancaires (comptes courants, Livrets A, PEL, …) Le modèle à mettre en œuvre contiendra au moins deux structures différentes : *User_Account* et *Account* sachant qu'un titulaire peut posséder plusieurs comptes et un compte peut avoir deux titulaires au maximum (le compte joint sur le compte courant seulement).

Les données des utilisateurs seront stockées dans des fichiers au format **JSON** [1] ainsi qu'un fichier nommé *listUser.dat* servant d'annuaire des utilisateurs.

Les données des comptes seront, elles, stockées dans des fichiers au format **CSV** [2] ainsi qu'un fichier, aussi, nommé *listAccount.dat* servant d'annuaire des comptes.

[1] JSON : https://fr.wikipedia.org/wiki/JavaScript_Object_Notation
[2] CSV : https://fr.wikipedia.org/wiki/Comma-separated_values

« Tatan, elle fait du flan ! »

- Kadoc

Démarche du projet

D'un accord commun, nous avons décidé de réaliser le projet en utilisant **Visual Studio Code** [3] afin de pouvoir ecrire du code (Visual Studio Code proposant une fonctionnalité *IntelliSense – Intelligent Code Completion*) ainsi que le **Bash Ubuntu pour Windows 10** [4] proposant un environnement Linux sous Windows.

Afin de gerer les versions et travailler ensemble, nous avons choisit d'utiliser **GitHub**[5], service que nous avions déjà utilisé pour un precedent projet.

Enfin, pour communiquer entre nous, c'est principalement **Messenger** [6] qui a été utilisé (quand la communication directe était impossible).

Le nom *KreGit*, jeu de mot entre la banque turque *Yapi Kredi Bankasi* [7]et GitHub est le nom de la banque (fictive) de notre projet.

[3] Visual Studio Code : https://code.visualstudio.com/
[4] Bash Ubuntu : https://korben.info/installer-shell-bash-linux-windows-10.html
[5] GitHub : https://github.com/
[6] Messenger : https://www.messenger.com/
[7] Yapi Kredi Bankasi : https://en.wikipedia.org/wiki/Yap%C4%B1_ve_Kredi_Bankas%C4%B1

Mode d’emploi

Le téléchargement du programme se fait sur la page GitHub [8]du projet ou bien en écrivant

```
git clone https://github.com/ctruillet/KreGit.git
```

dans votre terminal favori (terminal Linux préféré).

Une compilation du programme se fait en écrivant `make all` ou `make run` si le programme a déjà été compilé precedement.

[8] Page GitHub : https://ctruillet.github.io/KreGit/

« Moi j'ai appris à lire, ben je souhaite ça à personne. »

- Léodagan

Organisation du programme

Découpage et rôle des fonctions

Afin de rendre plus facile la conception du projet, nous avons décidé de séparer le code, au départ, en cinq librairies agissantes chacune sur des aspects bien spécifiques du projet.

Notre choix de stocker les données du l'utilisateur en format JSON nous a conduit à utiliser une librairie extérieure nommée Parson nous permettant d'écrire et de lire un fichier JSON de manière relativement facile.

Main

Composée d'une seule fonction du même nom, elle est, sans surprise, constituée de deux phases : l'initialisation, l'exécution.

Là où la première phase ne fait « que » créer le fichier .log permettant de suivre l'exécution, la deuxième phase est un peu plus complexe.

Afin de gérer au mieux les états possibles à tout moment de l'exécution, il a été choisi d'utiliser une machine à états.

	Title	Connect	Create_ User	Create_ Account	Remove_ Account	Admin	Customer	Show_ ListAccount	Show_ Account	Transfer	Change_ Pwd	Info	End
Title		X	X									X	X
Connect						X	X						
Create_ User							X						
Create_ Account							X						
Remove_ Account						X	X						
Admin	X							X			X		X
Customer	X			X				X			X		X
Show_ ListAccount						X	X		X				
Show_ Account						X	X			X			
Transfer						X	X						
Change_ Pwd						X	X						
Info	X												
End													

Figure 1 : Matrice des Etats

« Arthour pas changer assiette pour fromache !»

- Le Burgonde

Display

Conscients de la nécessité d'interaction entre l'utilisateur et le programme, nous avons décidé de regrouper toutes les fonctions dans cette librairie.

Ainsi, nous pouvons y trouver les fonctions de création de nouvel utilisateur et de nouveau compte (chacune respectivement liée aux fonctions de la librairie User_account et Account) proposant à l'utilisateur un formulaire

Outre ces deux fonctions, sont aussi présentes les fonctions *connect* et *deconnect* qui, comme leurs noms l'indique permettent à l'utilisateur de se connecter (récupérer le nom, prénom, mot de passe de l'utilisateur, le chiffrer afin de le comparer avec celui stocké, récupérer le fichier JSON associé à partir du fichier *listUser.dat* et enfin charger les informations dans la structure *User_account*) et de se déconnecter.

Afin de consulter les comptes, deux fonctions s'ajoutent à la liste : *displayListAccount* et *displayAccount* qui affiche les comptes et permet leurs consultation (*cf. figure 2*). Ces fonctions permettent aussi l'accès au menu donnant la possibilité de faire une nouvelle opération ou de supprimer le compte.

Aussi, afin de pouvoir naviguer entre les différents états du programme, une fonction *nav* a été conçue. Son fonctionnement est plutôt simple, à partir de l'état dans lequel l'utilisateur est, elle affiche le menu correspondant et permet d'aller uniquement vers ses états.

Enfin, elle comporte d'autres « petites » fonctions telle que *error* qui … affiche une erreur ou *kaamelott* qui affiche une citation aléatoire de la série Kaamelott[9].

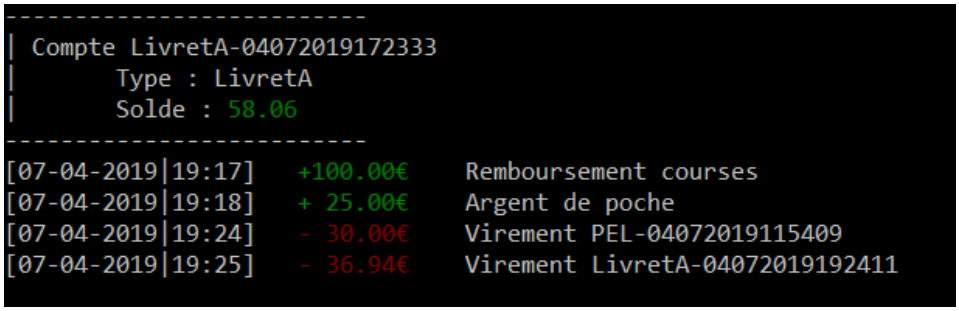


Figure 2 : Affichage d'un compte

[9] Kaamelott : https://fr.wikipedia.org/wiki/Kaamelott

« C'est pas faux ! »

- Perceval

User_account

Ayant eu pour consigne d'utiliser au moins deux structures, il nous a semblé évident qu'il faille créer une librairie pour chacune d'elles. Ainsi la librairie User_account était née.

Tout d'abord, elle définit un pointeur sur la structure User_account (qui contient l'identifiant (unique et généré par la fonction createUserID), le nom, le prénom et le mot de passé (chiffré) de l'utilisateur ainsi qu'un entier *isAdmin* (1 si l'utilisateur est administrateur) et un pointeur vers une structure *Account* que nous verrons par la suite).

Outre les nombreux getters et setters, d'autres fonctions accompagnent cette librairie.

Tout d'abord, il y a *create_user_account* et *charge_user_account* qui s'occupent de créer un nouveau compte (donc de créer un fichier JSON et de le remplir ainsi que créer un *User_account* qui va pouvoir être utiliser pendant toute l'exécution et enfin d'ajouter ce compte à *listUser.dat*) et de charger un compte déjà existant.

Et enfin, nous avons la fonction *changePwd* qui permet à l'utilisateur de changer de mot de passe. La structure du programme et la façon dont le langage C est conçu nous a amené à la nécessité d'exécuter les opérations suivantes : chiffrer le mot de passe, effacer l'utilisateur dans listUser.dat, recréer un fichier JSON sur l'ancien déjà existant (la modification d'une seule ligne étant impossible simplement).


```
{
    "user_account": {
        "ID": "04062019190645",
        "admin": 0,
        "firstname": "Clement",
        "name": "Truillet",
        "pwd": "45VWJbfUGfRqM",
        "List_account": [
            "PEL-04072019115409",
            "LivretA-04072019192411",
            "PEL-04072019192417"
        ]
    }
}
```


Figure 3 : Fichier JSON d'un utilisateur

« Elle est où la poulette ?»

- Kadoc

Account

De la même façon qu'avec la librairie *User_account*, la librairie *Account* gère les interactions avec la structure *account* (qui contient l'identifiant du compte (unique et généré par *createAccountID*), son type et un pointeur vers le compte suivant).

La structure *Account* étant en fait une liste chainée de compte, il est assez simple de la manipuler (la structure *User_account* pointe vers le début de la liste et un compte pointant vers *NULL* signifie qu'il est le dernier de la liste). Néanmoins, pour ce projet, il a fallu mettre en place des fonctions un poil plus atypique telle que *List_accountToString* qui renvoie la liste des comptes sous forme de chaine de caractère afin de la stocker de manière correcte dans le fichier JSON et *nbrAccount* qui compte le nombre de comptes appartenant à un utilisateur.

Aussi, il a fallu écrire des fonctions manipulant le fichier CSV lié au compte telles que *createAccountCsv* (qui crée le fichier et l'ajoute à listAccount.dat) et *getSolde* qui renvoie le solde disponible d'un compte.

Enfin, pour permettre la suppression d'un compte, deux fonctions ont été requises, l'une nommée *removeAccount* qui supprime le compte dans la liste chainée et supprime le fichier CSV lié au compte, et l'autre *removeAccountInList* qui supprime le compte dans le fichier *listAccount.dat.*

```
date,operation,solde,comments,
[07-04-2019|17:23],0.00,0.00,,
[07-04-2019|19:17],100.00,100.00,Remboursement courses,
[07-04-2019|19:18],25.00,125.00,Argent de poche,
[07-04-2019|19:24],-30.00,95.00,Virement PEL-04072019115409,
[07-04-2019|19:25],-36.94,58.06,Virement LivretA-04072019192411,
```

Figure 4 : Fichier CSV d'un compte

Organisation du programme

« CUUUUUUUUU ILLÈÈÈÈÈÈRE !

- Le Burgonde

Log

Issue d'un projet antérieure, cette fonction n'a pas changé et a pour seul but que d'écrire dans le fichier log généré à chaque exécution du programme.

Il est donc composé de deux fonctions, l'une qui créer le fichier (avec un nom unique) et l'autre qui écrit dedans.

Bien qu'inutile pour un utilisateur lambda, elle est extrêmement importante dans les phases de débogage du programme.

Encrypt

Au départ, cette librairie devant être chargée de chiffrer une chaine de caractère par l'algorithme md5. Malheureusement, à cause d'un manque de temps et d'un bug aujourd'hui toujours non résolu, il a été entendu de l'abandonner à profit d'une fonction obscure nommée *crypt* qui chiffre une chaine de caractère de manière suffisante pour le projet.

Parson [10]

Afin de manipuler les fichiers JSON, il était hors de question de programmer nous-même les fonctions nécessaires (par manque de temps et par envie de comprendre comment l'utilisation de librairie externe se fait dans le cadre d'un projet).

C'est donc par cette librairie que les entrées/sorties avec tous les fichiers JSON se fait.

[10] Parson : https://kgabis.github.io/parson/

« C'est systématiquement débile, mais c'est toujours inattendu »

- Arthur

Bilan

En conclusion, ce projet a permis d'atteindre des limites.

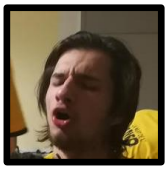
Une dead-line tout d'abord, heuresement repoussée mais qui a fait monter une pression accelerant alors la production de code.

Une exploration des limites du cahier des charges qui a induit une supressions de certains points jugés absurdes (demande de création de compte à l'administrateur) mais aussi à l'ajout de certains points (manipulation de fichier JSON non imposée, easters eggs).

En somme, c'était un projet compliqué non pas par les tâches a effectuer mais par le nombre de choses nouvelles a assimiler (compréhention du fonctionnement de librairie externe, du format JSON et CSV).

Enfin, ce projet nous a permis d'apprendre les bases du travail collectif (répartitions des tâches et entraide) sur un projet informatique. C'était un bon moyen de confronter nos acquis avec un projet ayant une finalité, ce qui est plus motivant que de simples exercices et les difficultés rencontrées nous ont permis de prendre de l'expérience et des habitudes de travail et de réflexions pour éviter ces problèmes dans un futur plus ou moins proche.

Informations de contact



Valentin Frydrychowski
Commit-man
valentin.frydrychowski@univ-tlse3.fr

Clément Truillet
Chef du monde incontesté
clement.truillet@univ-tlse3.fr

Informations sur l'entreprise

Kregit
https://ctruillet.github.io/KreGit/