Blaise Lucas
Chagras Flavien
Kesseiri Mohamed
Mayer Gauthier
Thommet Sacha

S13 - Porte monnaie virtuels

Années 2019 - 2020

Tuteur : Pierre-André Guénégo

Suivi : Dominique Lechaudel

Sommaire

1. Introduction

Ce document est une synthèse présentant l'évolution de notre projet tutoré au fil des différentes itérations. Il sera d'abord composé d'une présentation du projet et de l'équipe, puis de l'analyse qui a été faite à savoir le découpage des différentes fonctionnalités qui ont été faites, la conception de ces fonctionnalités et surtout les différences entre les idées et la conception de départ et le résultat final. Ensuite, nous présenterons les différentes fonctionnalités décrites juste avant pour enfin conclure par ce que nous a apporté le projet au niveau des compétences techniques et des méthodes de travail.

a. Présentation du projet

Le projet consiste à mettre en place un système de "porte-monnaie virtuels" indépendants d'un système bancaire existant, sécurisés et adaptés à différents types d'établissements (commerces locaux, hôtels, hôpitaux avec des personnes à responsabilités réduites…). Le système comprendra donc des outils bancaires comme des terminaux de paiement électronique et des cartes "bancaires" spéciales liées à ce système, ainsi qu'une application web permettant la gestion des portes-monnaie comprenant notamment un suivi des transactions, la possibilité d'ajouter des fonds, un système d'épargne propre au système etc. Le système se veut flexible et saura s'adapter à l'utilisation que l'opérateur veut en faire : par exemple si un cinéma utilise ce système, il sera plus intéressant d'avoir des places de cinéma sur la carte plutôt qu'une somme d'argent. L'utilisation de la carte sera donc adaptée au besoin de

l'établissement qui l'utilise, ils ne seront pas simplement limités par une carte qui ne peut contenir que de l'argent.

b. Présentation des rôles de l'équipe

Application Web :

Flavien Chagras

API et transactions :

Gauthier Mayer

Mohammed Kesseiri

Serveur TPE :

Sacha Thommet

TPE :

Lucas Blaise

Sacha Thommet

c. Planning de déroulement

Date		
29/09 au 09/10	Mise en place des groupes + Choix des sujets	Mail envoyer avec nom des personnes du groupe et le choix du sujet
15 /10 au 11/11	Faire une étude de l'existant et s'informer	Document à envoyer avec toutes les informations récoltés sur notre sujet
12/11 au 9/12	Etude préalable	Document pdf qui présente le sujet, les fonctionnalités et différents diagrammes.
9/12 au 11/01	Premiere iteration	- Réalisation de la BDD, - Configuration du Raspberry, - Squelette du site + page de connection + possibilité de faire des virements entre utilisateurs
20/01 au 8/03	Seconde itération	- Documentation (Symfony, easy admin Bundle, Arduino), - Mise en place du serveur + connection TPE-Serveur, - Possibilité de faire un virement avec une carte depuis le TPE - Site : finalisation de l'espace admin (nom sur les virements, bilan du moi, graphique du solde)
9/03 au 3/04	Dernière itération	- Mise en place d'une API, - Site : espace admin, mise en place du système de blockchain

Ce tableau sert de résumé, si vous voulez des informations plus détaillées (réunion, lien de documentation, fonctionnalité...) nous avons un trello :

https://trello.com/b/PFbZMaMn/s3as13blaisechagraskesseirimayerthommet

2. Analyse

a. Découpage fonctionnel

Le système se décompose en deux parties : Un partie physique et une partie virtuelle.
La partie virtuelle se compose d'une application web qui sera également exportée sur mobile.
Pour l'application, nous aurons, par catégorie d'utilisateurs:

Les clients qui auront la possibilité:

- D'effectuer des virements vers un autre comptes
- de demander un virement
- de visualiser des statistiques et un historique du compte (graphiques, relevés etc)
- de bloquer sa carte

L'opérateur sera la personne en charge du système et de son paramétrage, il pourra:

- Créer des comptes
- Assigner une carte à un compte
- Supprimer une carte et un compte
- Bloquer une carte
- Modifier paramètres du système (bloquer virement)

La boutique aura les mêmes droits que les clients mais aura la possibilité d'assigner un ou plusieurs TPE à son compte contrairement aux clients.

Il est important de souligner que les clients et les boutiques auront **tous deux un compte "utilisateur"** alors que l'**opérateur aura un compte d'administration**.

Pour ce qui est de la partie physique, nous aurons les TPE, ils feront le lien avec le serveur web qui se chargera de vérifier si le code est correct et si l'utilisateur à les fonds nécessaires au paiement à l'aide d'une API.

Pour mieux comprendre le fonctionnement des différentes fonctionnalités nous avons fait des diagrammes de séquence ainsi que des diagrammes d'activité.

Cas d'utilisation global du système :

Un utilisateur demande la création d'un compte. La création est effectuée par l'opérateur qui, en fonction de la demande, définira si cet utilisateur est une boutique ou un client. L'opérateur aura également la possibilité d'associer une carte au compte, cette carte doit permettre à l'utilisateur d'effectuer les paiements. Si le nouvel utilisateur est une boutique, il aura la possibilité d'associer un ou plusieurs TPE à son compte, lui permettant ainsi de recevoir un paiement par carte.

Voici le diagramme de séquence représentant un virement entre deux utilisateurs. Les critères de réussites sont plutôt clairs : l'utilisateur qui envoie les crédits doit avoir le bon montant débité et celui les recevant doit avoir le bon montant ajouté.

Voici le diagramme de séquence représentant un paiement par carte à l'aide d'un TPE. Dans ce scénario, la connexion du TPE au serveur et le paiement avec la carte sont tous les deux réussis. Le TPE ne sert que d'intermédiaire entre le serveur et l'utilisateur, il va envoyer les informations aux serveurs qui fera toutes les vérifications nécessaires. Pour que le paiement soit considéré comme réussi, il faut que l'utilisateur avec la carte soit débité du bon montant si il le possède sur son compte ou que le paiement soit refusé le cas échéant et que l'utilisateur du TPE reçoive les crédits sur son compte.

Voici le diagramme d'activité d'un paiement avec une carte

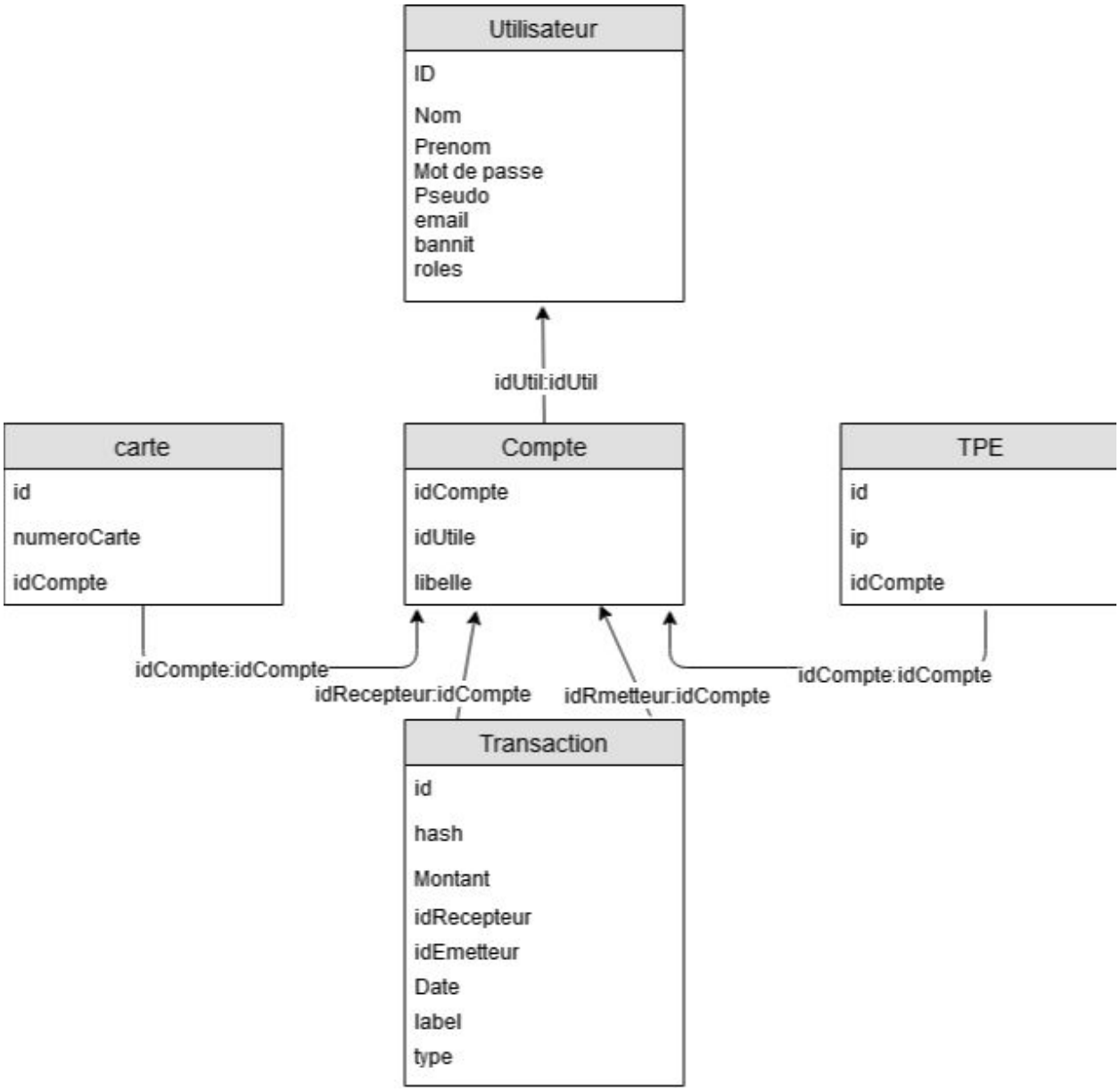
Diagramme d’état d’une carte lors d’un paiement

Diagramme de séquence expliquant comment le système calcule le solde d’un compte

Diagramme d’état d’association d’une carte à un compte

b. Schémas de conception

Schéma UML de la base de donnée :



c. Evolution par rapport à l'étude préalable

Une étude préalable a été faite avant de commencer réellement notre projet, cette étude nous a permis d'analyser le projet en lui même et ainsi de préparer sa mise en œuvre.

Pendant cette étape nous avons constaté que la base de données était la chose la plus importante pour la suite du projet, donc nous l'avons établi tous ensemble, ce qui était une bonne chose car nous n'avons pas eu à modifier l'architecture globale.

Ensuite, nous avons établi une liste de fonctionnalités pour le site web, nous avons réussie à mettre en place :

- un système de connection,
- possibilité de faire des virements sécurisé,
- une partie admin pour gérer le système (ajout de carte a un compte, ajout de TPE a une boutique, bloquer un compte,inscription d'un utilisateur),
- un graphique du solde actualisé en temps réel,
- un historique du compte.

Mais nous n'avons pas réussi à respecter la totalité de ces fonctionnalités comme par exemple le changement de devise n'a pas abouti. Nous n'aurons pas d'application mobile mais notre site est totalement responsive donc nous pouvons y accéder tout de même via un téléphone.

Pour finir, nous nous sommes penchés sur la sécurité du site, ce qui est la chose la plus importante pour un site qui propose des transactions. Dans un premier temps nous pensions avoir réussis cette étape avec une mise en place d'un serveur qui communique avec la base de données. Mais suite à une discussion avec notre tuteur,

nous avons constaté que cela ne suffisait pas. Il nous a donc lancé sur une piste qui été de faire une API pour éviter des injections sql dans la base.

Enfin pour garantir l'intégrité des transactions, chaque transaction possède un hash unique qui dépend de la transaction précédente qui garantit le fait qu'aucune transaction ne peut être modifiée. On utilisera un système de blockchain où chaque transaction correspond à un bloc. Ainsi, à chaque nouvelle transaction, on vérifie le hash de la précédente afin d'être sûr de l'intégrité de la table.

Le modèle de hachage est représenté ainsi :

Soit x une transaction :

$$hash(x) = md5\ (\ hash(x-1) + infos\ (x)\ )$$

$$avec\ hash(x0) = md5\ (\ infos(x0)\ )$$

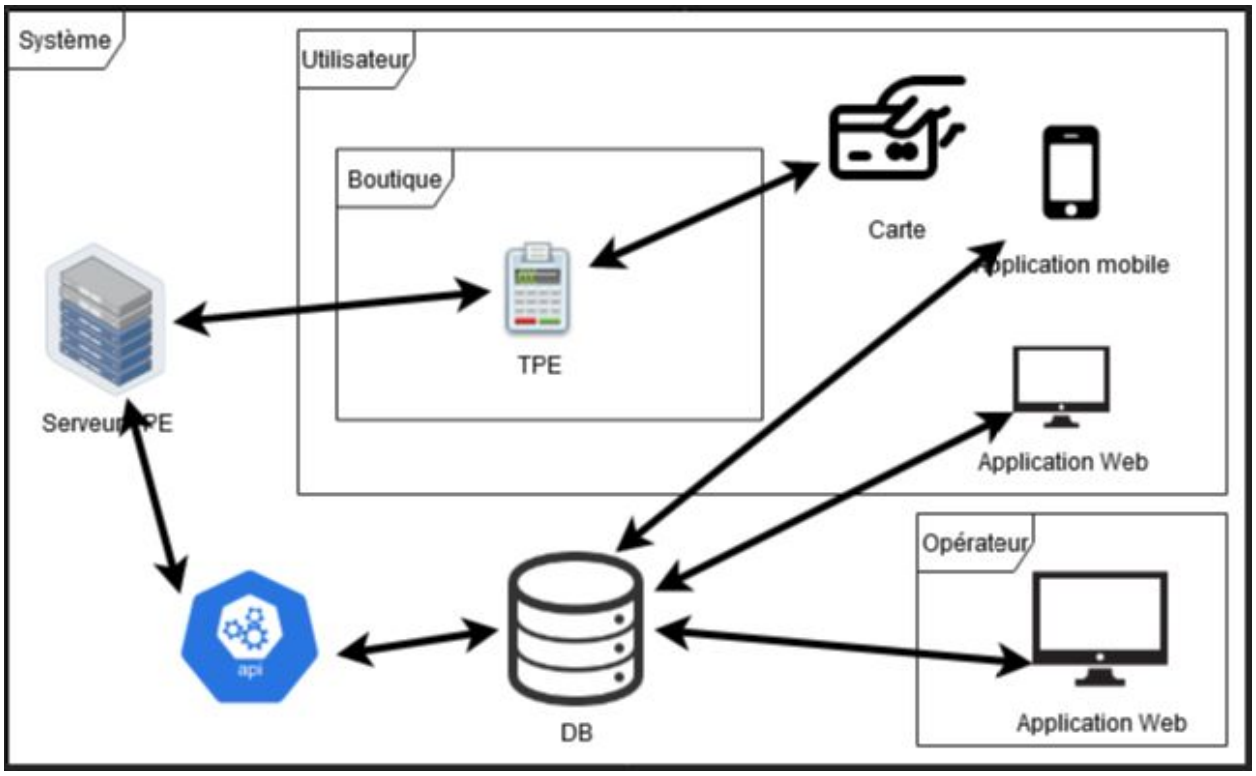
Infos(x) correspond à la concaténation des informations de la transaction x (montant, receveur, émetteur ...).

La sécurité de la communication entre les TPE et le serveur est tout autant cruciale. Nous avons utilisé le protocole SSL pour empêcher les utilisateurs malveillants d'intercepter des requêtes de paiement en cours à l'aide d'une attaque "Man in the middle". De plus un système de "Whitelist" permet de mémoriser les TPE appartenant au système (avec l'adresse IP). Seulement ces derniers peuvent communiquer avec le serveur. Finalement le démarrage du serveur nécessite un fichier de configuration contenant les identifiants d'un compte de l'application web. Ce compte doit en plus avoir un rôle permettant d'utiliser l'API.

3. Réalisation

a. Architecture logicielle

Schéma récapitulatif du système et des interactions :



- Application Web:

  Pour créer l'application web, nous avons opté pour le framework symfony 4.

  Symfony 4, nous permet d'utiliser le modèle MVC, ainsi chaque table dans la base de données a une classe. Nous suivons également le principe d'Active Record implémenté de base dans Symfony. De plus, Symfony contient de base un ensemble de Bundles pouvant être extrêmement utiles.

Nous avons utilisé l'ORM de Symfony: Doctrine. Ce dernier nous permet de gérer l'ensemble des intéractions avec la base de données.

Le Back-End a été fait en PHP et pour le Front-End, nous avons utilisé le moteur de templates Twig.

Coté administration, nous avons utilisé le composant EasyAdmin Bundle de Symfony 4, il permet de générer un espace administration entièrement configurable.

Concernant la sécurité, le composant sécurity de Symfony a été utilisé. Il permet d'attribuer un rôle aux utilisateurs et ainsi d'autoriser ou non un utilisateur à accéder à une page. De plus ce composant permet de contrôler les entrées dans des formulaires générés et ainsi de sécuriser ces derniers.

Pour le CSS du site, nous avons utilisé le template bootstrap SB Admin 2. Il convenait parfaitement pour la gestion d'un système bancaire. Nous avons également utilisé Datatable afin de styliser les différents tableaux.

- API : Pour créer l'API qui permet au serveur de communiquer avec la base de données sans l'exporter, nous avons utilisé une architecture d'**API SOAP** qui permet à différents composants d'un système utilisant notamment des langages de programmation différents de communiquer ensemble et donc de les rendre compatibles.

- Serveur TPE : Le serveur a été crée en Java, il associe à chaque TPE un thread "listener". Plusieurs TPE peuvent donc se connecter en même temps et effectuer des requêtes **parallèlement**. Il utilise une librairie permettant d'effectuer des

requêtes HTTP, pour communiquer avec l'API. Cette architecture permet d'améliorer la **sécurité** de la base de données, en ne permettant pas au serveur de communiquer directement avec cette dernière.

b. Difficultés rencontrées

- Application Web :

  L'utilisation d'un framework aussi complexe que Symfony 4 a impliqué de longues heures d'apprentissage. Nous avons opté pour des outils qui n'ont pas été vus en cours et par conséquent nous avons dû apprendre à utiliser ces derniers. Il n'y a pas eu de problèmes en particuliers. Il fallait simplement réussir à trouver comment le faire en se documentant.

- Serveur TPE :

  2 principales difficultées ont été rencontrées. Le serveur devait pouvoir communiquer avec plusieurs TPE simultanément. Pour répondre à cette problématique nous avons choisi d'allouer un thread par connexion entrante. Chaque thread utilise une instance d'un "listener" (qui écoute et répond aux requêtes d'un TPE différent), pour éviter les conflits dus à l'environnement multi-thread. Ensuite au départ le serveur communiquait directement avec la base de donnée, ce qui posait des problèmes de sécurité. Pour combler à cette faille, nous avons mit en place une API, et de ce fait on a dû adapter le serveur pour qu'il utilise celle-ci.

- TPE : L'écran et le clavier numérique n'étant pas compatibles avec l'ESP8266, nous avons dû créer un logiciel qui sert d'extension à l'ESP. Le logiciel envoie les données saisies directement à la carte programmable via les ports COM.

4. Conclusion

Nous n'avons malheureusement pas réussi à finir l’ensemble des fonctionnalités imaginées initialement. Cependant notre projet peut être repris par une autre équipe, nous avons documenté le code, respecté les conventions déjà mise en place. Nous nous sommes assurés de ne pas ajouter de fonctionnalités non terminées, par conséquent si ce projet est récupéré, il sera simple d’implémenter de nouvelles fonctionnalités sans entrer en collision avec du code déjà existant.

5. Annexes

Trello : https://trello.com/b/PFbZMaMn/s3as13blaisechagraskesseirimayerthommet

Git :

https://bitbucket.org/depinfoens/s3a_s13_blaise_chagras_kesseiri_mayer_thommet/src/master/

Documentation Symphony :

https://symfony.com/doc/current/security.html

Documentation Easy Admin Bundle:

https://symfony.com/doc/current/bundles/EasyAdminBundle/index.html

Documentation Composant Security:

https://symfony.com/doc/current/security.html

Documentation Doctrine:

https://symfony.com/doc/current/doctrine.html

Documentation DataTable:

https://www.datatables.net/manual/

SB Admin 2:

https://github.com/BlackrockDigital/startbootstrap-sb-admin-2

Bootstrap:

https://getbootstrap.com/