Trésorerie d'ADH6

- Premier livrable -

Julie Chevrier François Horta
Gabriel Prévault Adrien Triquet

Encadrant : Olivier Berger

10 février 2019

Table des matières

ADH6 est la future interface de gestion des adhérents de l'association MiNET. Actuellement, l'association utilise la cinquième version de cette interface, ADH5.

ADH5, et bientôt ADH6, permet notamment de créer des comptes aux adhérents, de les faire cotiser afin de leur fournir un accès Internet, de gérer les adresses MAC des appareils des adhérents, configurer les ports ethernet du local MiNET ou encore, dans une moindre mesure, d'avoir accès à la trésorerie de l'association.

ADH6 est déjà en développement par certains membres de l'association, mais la partie de la gestion de la trésorerie n'est pas encore implémentée, c'est pourquoi nous avons décidé de nous y atteler dans le cadre du projet de développement informatique.

I. Analyse des besoins

1. Produit du projet

Création de la trésorerie sur ADH6.

2. Fonctions du produit

Ci-après la liste des fonctionnalités que nous souhaitons implémenter, avec un ordre de priorité entre 1 et 5, 5 étant la priorité la plus élevée.

- Système de trésorerie à double entrée **(5)**
    - Gestion de la cotisation pour les adhérents de l'association MiNET et de la vente de produits (câbles ethernet, adaptateurs USB/Ethernet)
    - Historique des transactions
    - Création de comptes pour les clubs
    - Gestion de la clôture des comptes
- Génération du bilan comptable et du compte des résultats **(4)** pour chaque compte et sous-compte **(3)**
- Ajout automatique des factures **(3)**
    - Scan des factures et importation dans ADH6
- Reçu automatique **(3)**
- Système relié à la caisse **(3)**
- Gestion des dettes **(2)**
- Système relié au terminal de paiement électronique **(2)**

3. Systèmes d'acceptabilité et de réception

- Gestion des nombres sans approximation (gestion des nombres flottants)
- Interface claire
- Logiciel opérationnel pour le trésorier MiNET
- Sécurité de la base de données

4. Contraintes techniques

Interface web basée sur ADH6 utilisant déjà les technologies suivantes :

- Flask
- Angular
- Docker
- Swagger

Voici la liste des raisons pour lesquelles nous souhaitons développer cette solution nous-mêmes et non pas intégrer une solution toute faite :

- pour avoir un maximum de flexibilité
- les solutions existantes sont trop complexes, nous n'avons pas les mêmes besoins qu'une entreprise
- pour l'intégrer au mieux au sein de l'architecture existante
- pour adapter au mieux ce que l'on va faire aux réels besoins du trésorier MiNET

5. Clauses juridiques

ADH6 est distribué sous la licence GNU General Public License v3.0.

II. Spécification fonctionnelle générale

Remarque : les utilisateurs ayant un niveau de droits plus élevé héritent bien évidemment des droits de ceux ayant un niveau de droits plus faible.

Par exemple, un membre actif MiNET ou un utilisateur non privilégié aura donc les mêmes cas d'utilisation qu'un adhérent auxquels s'ajoutent ceux indiqués dans la Figure 2.

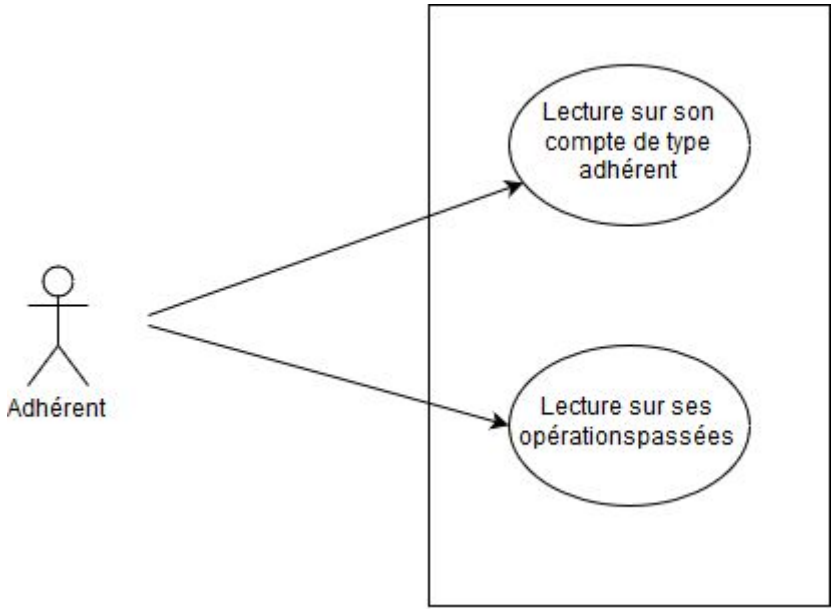


Figure 1 - Diagramme de cas d'utilisation pour le profil Adhérent

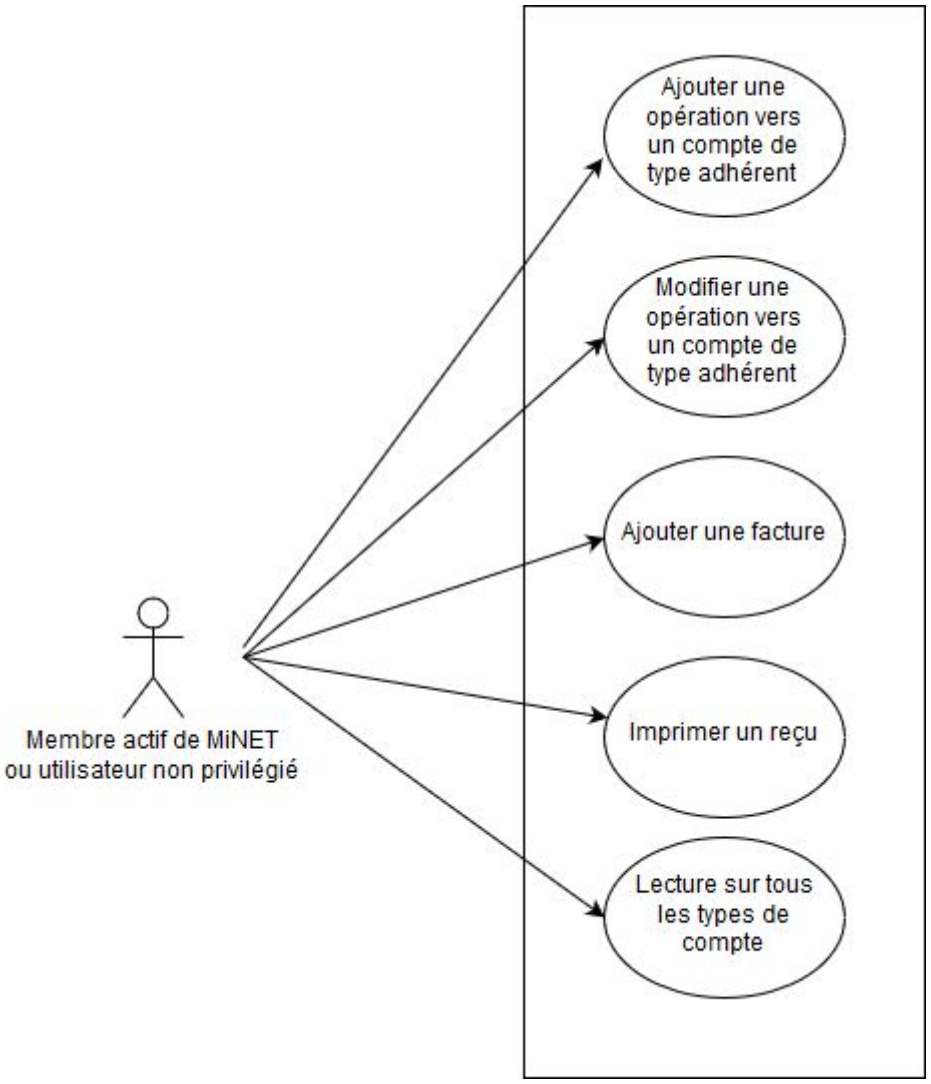


Figure 2 - Diagramme de cas d'utilisation pour le profil Membre actif MiNET ou Utilisateur non privilégié

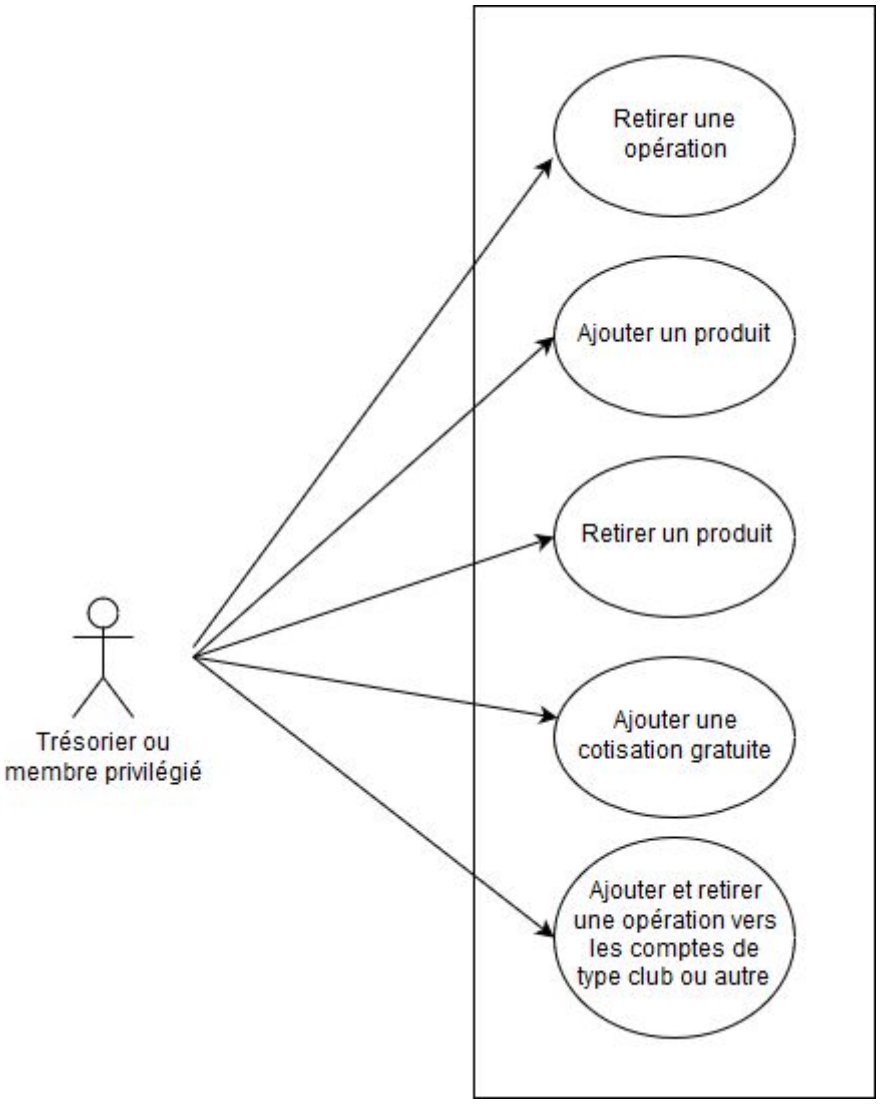
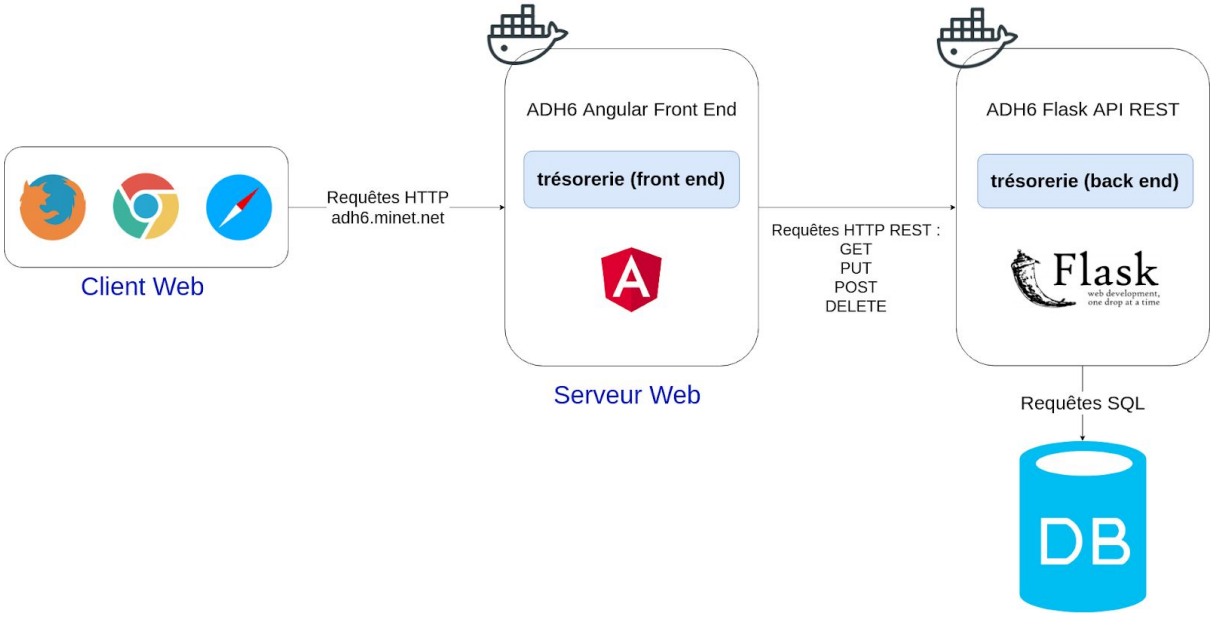


Figure 3 - Diagramme des cas d'utilisation pour le profil Trésorier ou Utilisateur privilégié

1. Architecture globale du projet



2. Côté Frontend

Nous utiliserons le framework AngularJS basé sur le langage de programmation Javascript qui permet la génération de pages web sur les terminaux clients. Cela nous permet ainsi d'obtenir un squelette déjà prédéfini pour notre application et donc de profiter de classes déjà implémentées par exemple.

3. Côté Backend

Nos fonctions reposeront sur le framework Flask, qui permet de faire du développement web en utilisant le langage de programmation Python.

3.1. Serveur-API

Pour imprimer les reçus automatiquement, nous aurons besoin d'une route /api/transaction/{id}/print. Pour ajouter une facture dans la base de données, nous utiliserons /api/compte/{id}/upload. Enfin, pour générer une opération, nous aurons besoin d'une route qui inscrit la transaction dans la table Opération à savoir /api/transaction/{id}/paiement.

Remarques :

- Les routes de l'API REST ont été spécifiées avec swagger.
- Une API REST est bien sûr orientée client-serveur.

3.2. Stockage permanent des informations

(Voir partie IV. pour le schéma SQL lié)

Il existe trois manières de stocker des informations de manière permanente (ie qui subsistent au redémarrage d'une machine) : un stockage dans un fichier avec utilisation de délimiteurs, les bases de données NOSQL et les bases de données SQL.

Pour des raisons de compatibilité avec l'application existante, nous avons choisi d'utiliser une base de données reposant sur le langage SQL.

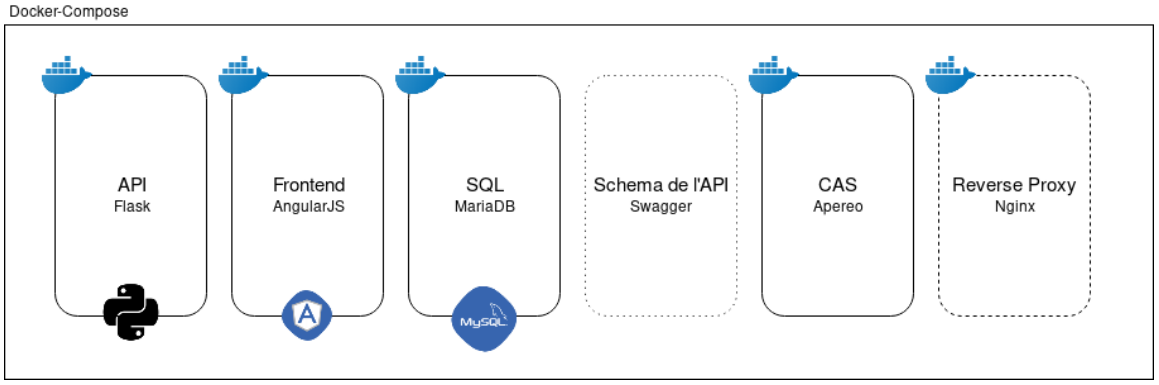
Pour gérer les différentes fonctionnalités, nous avons décidé de créer différentes tables dans la base de données. Nous reprenons la table *Adhérent* déjà implémentée sur l'ancienne version de ADH6 à savoir ADH5, celle-ci répondant déjà à nos besoins. Une table *Compte* où chaque compte sera caractérisé par son identifiant, son type : ce peut être le compte d'un club de MiNET, d'un adhérent ou autre (par exemple celui d'un événement). Une table *Produit* où chaque produit sera également caractérisé par un identifiant, son nom permettant de désigner sa nature ainsi que son prix d'achat et de vente. En effet, nous proposons à l'achat les différentes cotisations mais également des produits comme des câbles ethernet (3m et 5m) ainsi que des adaptateurs. Enfin, nous avons implémenté une table *Transaction* permettant de recenser toutes les opérations. Une transaction est caractérisée par son identifiant, son nom (cotisation, subvention, achat, ...), par le produit concerné, le montant de la transaction, le compte source et le compte destination, le moyen de paiement utilisé et les pièces jointes éventuelles (ceci nous permettra ainsi de joindre des factures).

Comme indiqué plus haut, les différentes opérations sur les bases de données seront donc effectuées dans le langage SQL. Nous exploiterons alors au mieux les fonctionnalités inhérentes au framework Flask pour réaliser ces requêtes SQL - il est notamment prévu d'utiliser SQLalchemy déjà utilisé par ailleurs dans ADH6.

Remarque générale : le web est basé sur un modèle client/serveur, les nouvelles applications webs exécutent de plus en plus de contenus sur les clients. Pour ce faire, les clients webs (navigateurs) exécutent du code (souvent Javascript ou WebAssembly) afin de générer un rendu plus dynamique. Ceci est réalisé en plus des langages de descriptions traditionnels (HTML pour la structure de la page et CSS pour le style/couleur/etc) qui sont toujours utilisés.

4. Automatisation du déploiement

L'intégralité des services est déployée en containers. L'orchestration est gérée par Docker Compose, ce qui permet un redéploiement rapide lors du développement et un environnement uniforme indépendant de l'architecture hôte.



5. Tests

Les tests unitaires passent par `pytest`. Ils sont lancés manuellement et automatiquement lors du déploiement. Les tests de validation seront établis manuellement en revenant à ce document si nécessaire.

III. Regroupement modulaire

- Action sur les transactions
  - Au TPE
  - À la caisse
  - Impression de reçus
  - Ajout des factures (imprimante / scan)
- (Gestion des dettes)
- Action sur les comptes
  - Bilan comptable et compte de résultats
  - Création, clôture, historique des différents comptes

Pour traduire en terme d'API, chaque bloc correspond respectivement aux endpoints `/transaction`, (`/compte_minet`) et `/compte`.

IV. Description du flux des données entre les modules

Sous la forme du schéma SQL, dont la définition régit très clairement la structure même du code sous-jacent.