Rapport d'architecture

Equipe A

DEAL Emma
LORCERY Morgane
MARINI Claire
MEHDI Khadidja
PRINCE Jules

POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Table des matières

Introduction

Le projet que nous allons réaliser durant ce semestre consiste à créer un système associé à une carte multi-fidélité. Ce système permettra aux utilisateurs du programme de pouvoir bénéficier d'offres de la part des commerçants en fonction de leurs habitudes de consommation et grâce à un système de points. Cela permettra aussi aux commerçants d'attirer les consommateurs grâce aux offres qu'ils pourront proposer eux-mêmes. Ce système disposera également de nombreuses fonctionnalités comme l'ajout de favoris, la récolte d'informations/statistiques ou encore ajouter une carte de crédit.
Pour implémenter ce système, nous avons fait plusieurs hypothèse concernant l'utilisation des fonctionnalités mas aussi concernant le rôle de chaque acteur du système.

Voici les hypothèses nous ayant permis de réaliser les diagrammes qui vont suivre:

- On suppose que lorsqu'un consommateur réalise un achat dans un magasin partenaire, il pose sa carte de fidélité sur un détecteur, ce qui permet que cette achat lui apporte des points et que l'achat soir enregistré sur la plateforme.
- Si le consommateur réalise le paiement avec sa carte de fidélité (rechargée grâce à sa carte bancaire), en plus d'enregistrer l'achat et mettre à jour les points, le paiement sera déduit du solde présent sur la carte.
- On suppose que tout les avantages physique (exemple: les cadeaux) ou les services (exemple: stationnement gratuit) sont stockés dans une même table "avantages". Les avantages ont un nombre de points requis et/ou un nombre d'utilisation déjà effectué.
- Pour profiter dun avantage, un utilisateur doit d'abord sélectionner l'avantage sur l'application. Pour que cette sélection soit effective, il faut que l'utilisateur ait suffisamment de points sur sa carte et/ou un nombre d'achats effectués assez élevé. Une fois la sélection effective, il peut se rendre en magasin pour récupérer l'avantage à l'aide du numéro d'identification de l'offre. S'il s'agit d'un avantage tel qu'une durée de stationnement gratuit, il pourra l'utiliser sur la plateforme directement.
- Chaque partenaire (commerce, société de transport, mairie, ...) sont réunis dans la table "partners". Ils possèdent tous une ou plusieurs sociétés. Une société est représentée par son nom, des horaires d'ouverture et son adresse. Elle possède également des offres propre à elle.
- Les comptes partenaires et leur(s) société(s) sont crées sur demande par les employés de la mairie. Ils pourront ensuite modifier les informations de la/les société(s)
- VFP = moyenne par jour sur une semaine (minimum de 2 achats par jour en moyenne)

Cas d'utilisation

Acteurs primaires :

- Consumer : il représente les clients de notre application qui souhaitent bénéficier de la carte multi-fidélité. Ils procèdent à l'inscription dans un premier temps, puis peuvent se connecter lorsqu'ils le souhaitent pour gérer leurs cartes, leur profil, bénéficier de leurs offres, se renseigner sur les commerces et profiter du stationnement gratuit.
- Partner : il représente les commerçants qui souhaitent participer au programme de carte multi-fidélité. Il peut réaliser plusieurs actions comme se connecter, ajouter et/ou modifier les informations sur leur commerce et les offres proposées. Après plusieurs clients, ils pourront avoir accès aux informations de ces derniers et aux statistiques de leur commerce.
- Mayor : il représente les employés à la mairie qui sont des "partner" mais avec plus de drots. Ils peuvent donc réaliser les mêmes actions mais ils peuvent aussi enregistrer des comptes de commerçant, contacter les clients pour envoyer des offres personnalisées ou des sondages,

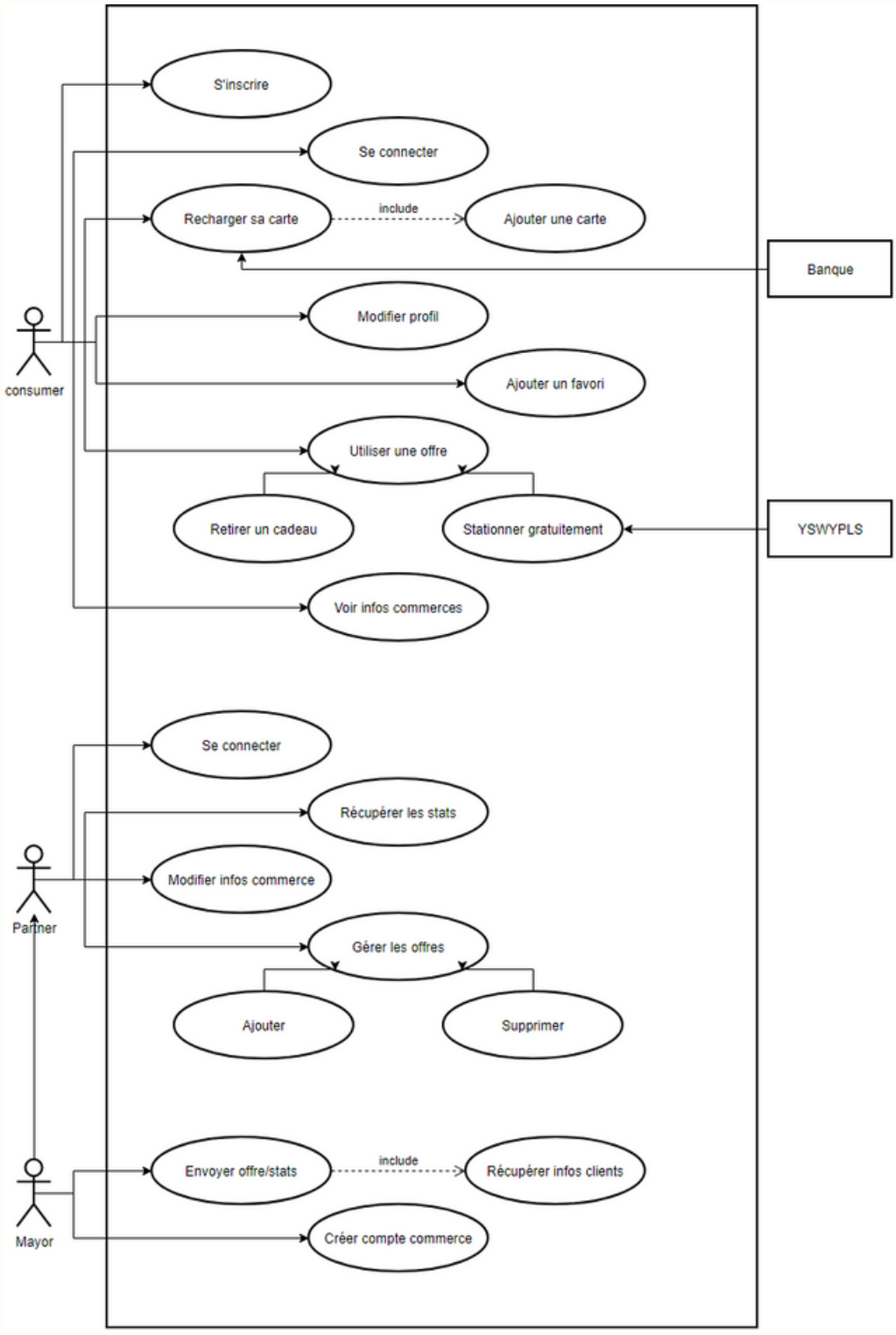
Acteurs secondaires :

- La Banque : c'est un acteur secondaire qui intervient seulement lors du rechargement de la carte d'un Consumer et peut retourner une erreur en cas d'échec de paiement et interrompre le processus de rechargement.
- I Know Where You Parked Last Summer : c'est un second acteur secondaire qui représente une plateforme externe qui va permettre de lancer un compteur afin de mesurer la durée d'un stationnement, répertorier ces minutes ainsi que les paiements avec la plaque d'immatriculation. Il peut aussi notifier le client dans le cas d'un stationnement gratuit pour une durée déterminée.

Découpage des cas :

A partir de la liste des acteurs primaires, nous avons pu réaliser le diagramme de cas d'utilisation suivant. Chaque fonctionnalité est représentée par un cas. Certains dépendent d'un autre cas, comme par exemple recharger sa carte qui nécessite d'avoir enregistrer une carte. Cette action va aussi nécessiter l'intervention d'un acteur secondaire (ici il s'agit de la banque). Quant au cas Gérer les offres par exemple, il peut se présenter de plusieurs manières : il est possible d'en ajouter ou d'en supprimer.

Diagramme de cas d'utilisation :



Objets métier

Consumer: Cette classe représente les consommateurs, les habitants ayant une carte multi fidélité. Elle possède comme attributs des informations sur le consommateur et est associé à une carte.

Card: Cette classe représente la carte multifidélité. Elle possède comme attributs son nombre de points, liés aux achats préalablement fait par le consumer, et de l'argent que ce dernier a viré dessus. Elle est associé à un unique consommateur.

Partner: Cette classe représente les commerçants. Elle possède comme attributs des informations sur ce dernier.

Society: Cette classe représente la société du partner. Par exemple, la société gérant les places de parking est une société du Maire.

Shop: C'est un type de société plus précis: les commerces. Ils ont donc des horaires d'ouvertures et une adresse qui pourront être visible sur l'application. Shop hérite de society.

Purchase: Cette classe représente un achat, avec son montant, la date et le magasin où il a été effectué, et la carte de fidélité lié. Cette classe nous permettre d'avoir un historique d'achats.

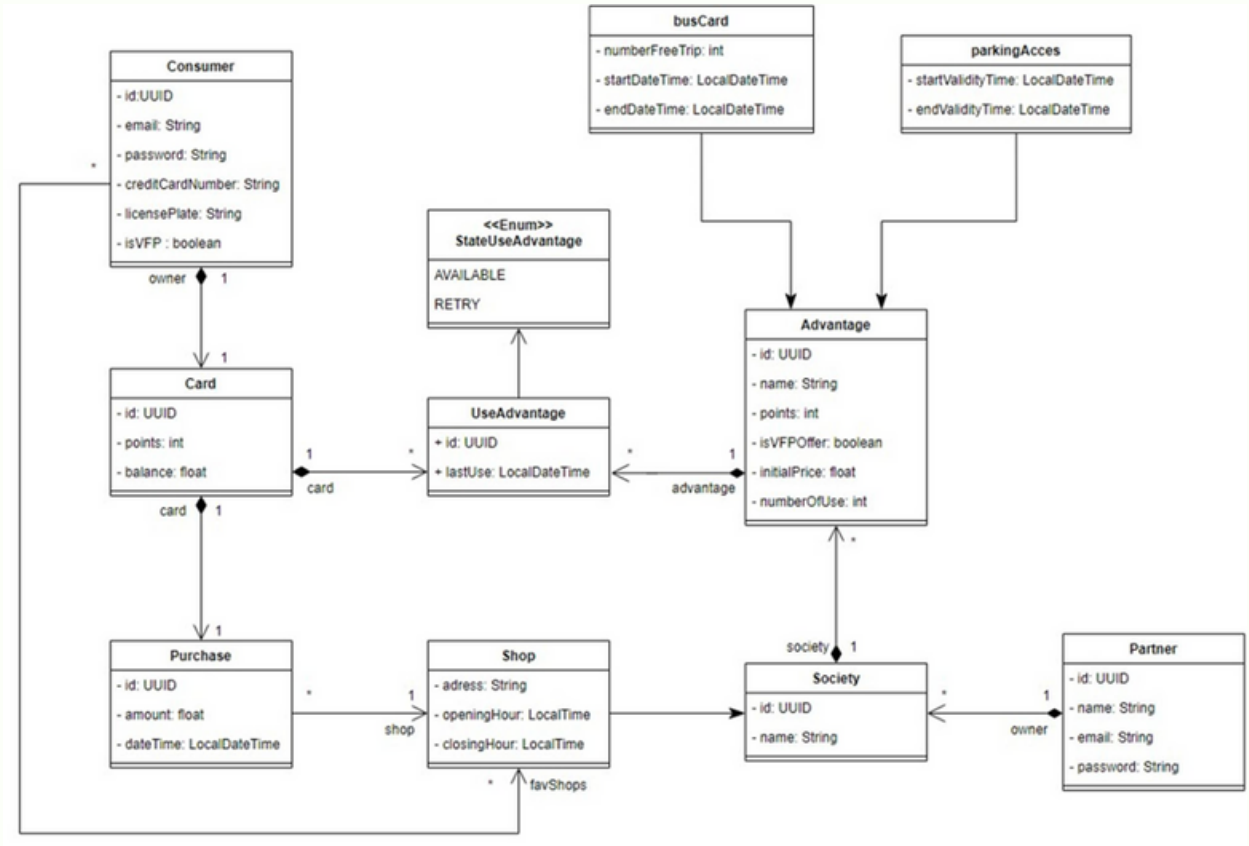
Advantage: Cette classe représente les offres ou les cadeaux que peuvent offrir les sociétés aux consommateurs. Elle possède comme attributs des informations sur l'offre comme son nombre de points associés mais aussi des attributs de statistiques, comme le nombre d'utilisation ou le montant initial qui nous sera utile pour vérifier que le magasin ne perd pas d'argent en proposant cette offre.

ParkingAccess et **BusCard** sont deux classes héritant d'advantage étant des offres plus spéciales, nécessitant notamment des calculs de temps.

AdvantageSelected: Ce sont les advantage ayant été sélectionnés ou récupérés par les consommateurs. L'attribut lastUse nous permettra de vérifier, par exemple pour une offre "Un ticket de bus offert par jour", si le ticket a déjà été utilisé dans la journée ou non. Cette classe nous permettra d'avoir un historique des avantages utilisés.

De cette manière, les responsabilités sont bien distribuées entre les classes en fonction du métier et chaque classe à connaissance des classes dont elle a besoin pour fonctionner et réaliser les fonctionnalités qui lui sont associées.

Diagramme de classe:



Interfaces

AccountModifier : Cette interface permet aux membres de la marie (Mayor) d'ajouter les comptes des partenaires (Partner) à la plateforme ainsi qu'ajouter leur entreprise associée (que ce soit une société proposant des services ou un magasin) .

```
public interface AccountModifier {

    Partner addPartnerAccount(Partner newPartner) throws AlreadyExistingAccountException;

    Society addSociety(Society newSociety);

}
```

AdvantageChoices : Cette interface permet aux consommateurs (Consumer) de sélectionner un avantage dont il veut profiter. Il pourra l'utiliser par la suite si c'est un avantage de type service .

```
public interface AdvantageChoices {

    SelectAdvantage selectAdvantage(Advantage advantage, Consumer consumer) throws NotEnoughPointsException;

    SelectAdvantage useAdvantage(Advantage advantage, Consumer consumer) throws IWYPLSConnexionException;

}
```

AdvantageSelectedManagement : Cette interface permet de créer un historique des avantages sélectionnée par les utilisateurs afin qu'ils puissent ensuite les utiliser si ce sont des service ou les récupérer en magasin si ce sont des cadeaux .

```
public interface AdvantageSelectedManagement {

    SelectAdvantage addSelectAdvantage(SelectAdvantage useAdvantage);
}
```

BalanceManagement : Cette interface permet de modifier le solde (balance) de la carte d'un consommateur et également de vérifier si le solde contient au moins un montant donnée .

```java
public interface BalanceManagement {

    float updateBalance(float money, Card card) throws BankRefusedPaymentException;

    boolean canBuy(float amount, Card card);
}
```

Bank : Cette interface permet de vérifier si un consommateur peut importer de l'argent de sa carte de crédit .

```java
public interface Bank {
    boolean importMoney(Consumer consumer, float value) ;

}
```

CardPayment : Cette interface permet aux consommateur de réaliser des achat dans des magasins partenaires avec leur carte, d'ainsi gagner des points. Le paiement peut être réaliser avec la carte elle même dans ce cas le solde sera décrémenté.

```java
public interface CardPayment {

    Purchase buyInShopWithFidelityCard(Card card, Shop shop, float amount) throws NotEnoughBalanceException;

    Purchase buyInShop(Card card, Shop shop, float amount) ;
}
```

CatalogModifier : Cette interface permet aux partenaires et au membre de la marie d'ajouter ou de supprimer des avantage sur la plateforme .

```java
public interface CatalogModifier {

    Advantage addAdvantage(Advantage advantage);

    void deleteAdvantage(Advantage advantage);
}
```

CollectGift : Cette interface permet aux partenaires possédant un magasin de stipuler qu'un cadeau sélectionné a été récupéré par un consommateur .

```java
public interface CollectGift {

    void giveGift(Advantage gift) ;
}
```

ConsumerInformationGetter : Permet d'évaluer si un consommateur est VFP a partir de la fréquence de ses achats hebdomadaires .

```
public interface ConsumerInformationGetter {

    boolean isVfp(Consumer consumer);

}
```

AdvantageCreator : Cette interface permet aux membres de la mairie de créer de nouveau avantage sur la platforme.

```
public interface AdvantageCreator {

    Advantage addAdvantage(Advantage advantage) ;
}
```

ConsomationInformationGiver : Cette interface permet aux membres de la mairie de récupère la listes des consommateur ainsi que de voir leur habitude de consommation en achat brut et en cadeaux recupérés.

```
public interface ConsomationInformationGiver {

   List<Consumer> getAllConsumer();

}
```

ConsumerModifier : Cette interface permet aux consommateur de modifié leur informations sur la plateforme , d'ajouter/retirer des favoris et également de remplir le solde de leur carte .

```
public interface ConsumerModifier {

    float reloadBalance(float amount, Consumer consumer) throws BankRefusedPaymentException;

    Consumer updateConsumer(Consumer consumer);

    Shop addFavShop(Shop shop, Consumer consumer);

    Shop deleteFavShop(Shop shop,Consumer consumer);

}
```

IsawWhereYouParkedLastSummer : Cette interface permet de mettre en relation le consommateur et l'application IsawWhereYouParkedLastSummer afin de demarer un compteur pour le temps de parking gratuit .

```
public interface IsawWhereYouParkedLastSummer {

    void startCounter(String licencePlate, LocalTime start,LocalTime end) ;
}
```

NotificationSender : Cette interface permet au propriétaire de magasin de notifier les consommateurs lorsqu'il change les heures d'ouverture de leur magasin , si ces dernier possèdent leur magasin en favoris .

```
public interface NotificationSender {

    void notifyConsumerNewSchedule(LocalTime schedule , Shop shop , ScheduleType scheduleType);
}
```

PointManagement : Cette interface permet de modifier le nombre de point d'une carte (ajouter/retirer de l'argent).

```
public interface PointManagement {

    int updatePoints(int numberOfPoints, Card card) throws NegativeQuantityException;

}
```

PurchaseManagement : Cette interface permet d'enregistrer l'historique de tout les paiements réalisés avec une carte de fidelité.

```
public interface PurchaseManagement {

    Purchase addSales(int numberOfPoints, Card card, Shop shop);
}
```

RegistryConsumerProcessor : Cette interface permet aux consommateurs de se connecter a la plateforme ainsi que de s'inscrire avec un email et un mot de passe .

```
public interface RegistryConsumerProcessor {

    Consumer login(String email , String password) throws AccountNotFoundException;

    Consumer signin(String email , String password) throws AlreadyExistingAccountException;
}
```

RegistryPartnerProcessor : Cette interface permet aux partenaires de se connecter a la plateforme .

```
public interface RegistryPartnerProcessor {

    Partner login(String email , String password) throws AccountNotFoundException;

}
```

ReminderSender: Cette interface permet aux membres de la mairie de notifier des consommateur avant qu'ils ne perdent leur statut VFP.

```
public interface ReminderSender {

    void reminderConsumer(Consumer consumer);
}
```

ShopModifier : Cette interface permet aux partenaires de la plateforme de modifier les information de leur magasin ainsi d'ajouter des avantages et en retirer .

```
public interface ShopModifier {

    Shop modifyShopInformation(Shop shop) ;

    Advantage addAdvantage(Advantage advantage);

    void deleteAdvantage(Advantage advantage);

}
```

ShopsInformationGetter : Cette interface permet a toute les personne connecté sur la plaforme de voir des informations sur les société présentes de la plateforme et leurs avantages. Si nous avons un compte consommateur Il permet également de voir nos magasin favoris et leur offre auquel nous sommes éligible avec notre nombre de points.

```
public interface ShopsInformationGetter {

    List<Shop> getAllShops();

    Shop getShopByName(String name);

    Shop getShopById(Shop shop);

    List<Advantage> getAllAdvantage();

    List<Shop> getFavShops(Consumer consumer);

    List<Advantage> getAllAdvantageEligible(Consumer consumer);

}
```

Composants

Commençons par les façades, qui sont au nombre de 7:

MayorAction : Ce composant façade est celui relié comme son nom l'indique aux actions effectuables par le maire. Il est le composant relié au plus de repository car il fait office d'administrateur étant donné qu'il s 'occupe de la création des comptes commerçants.

PartnerAction: Ce composant traite les actions réalisables par un partenaire commercial. Il a uniquement accès à son magasin en base de données. Il crée des avantages pour les clients et envoie des notifications aux clients quand ses horaires d'ouverture change.

ConsumerAction: Ce composant modélise les actions possibles du client. C'est la façade reliée au plus de composants car le client est au centre de notre système. Le consommateur a une carte qui lui est propre mais également des informations personnelles. Le client peut utiliser les fonctionnalités de parking en utilisant l'api **ISawWhereYouParkedLastSummer**.

PayementService : Ce composant va être appelé par le lecteur de cartes lors d'un passage en caisse. Il est celui qui va débiter le crédit présent sur la carte du client et lui attribuer des points en fonction des commerces.

ShowCase : Ce composant est la vitrine de tous les magasins disponibles en ville. Il permet aux clients d'avoir une vision d'ensemble sur tous les services que propose la ville.

StatisticsService : Ce composant permet de récupérer des informations de statistique sur l'utilisation du service dans notre ville.

On retrouve ensuite des composants qui font office de service :

Phone : Ce composant va avertir les utilisateurs ayant mis un magasin en favoris quand celui-ci change ses horaires, informations ou crée un nouvel avantage en boutique.

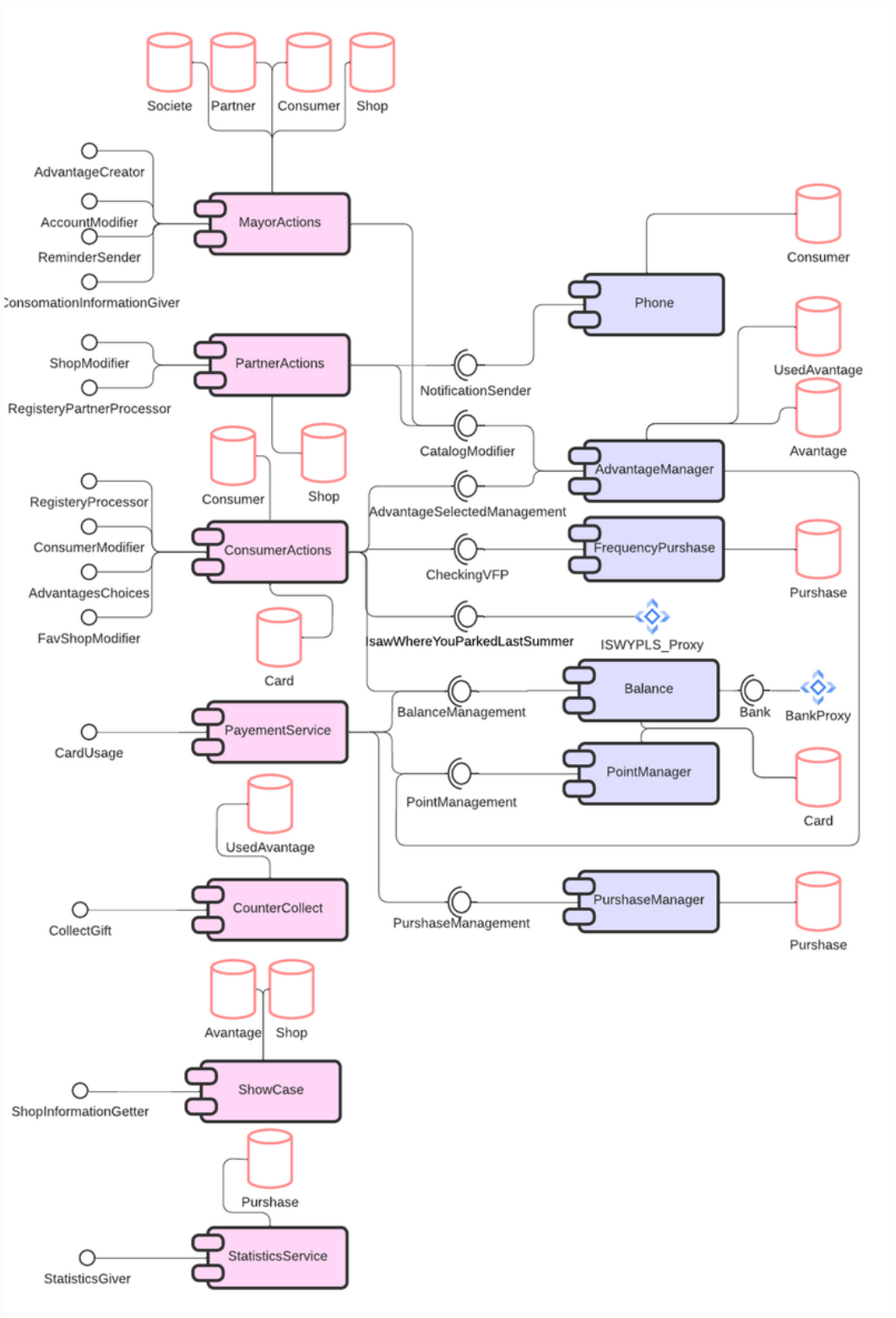
AdvantageManager : Ce composant se charge de prendre en main les avantages de toutes sortes. Il est appelé notamment quand le client utilise son avantage en boutique ou dans un autre service proposé par la municipalité. Il permet aussi de garder en mémoire tous les avantages utilisés.

FrequecyPurshase : Ce composant sert à vérifier si un client est VFP. Le calcul se fait en fonction de la fréquence hebdomadaire d'achat d'un client. À chaque achat, le service est appelé pour recalculer le score du client. A partir d'un certain palier le client devient **VFP**.

Balance : Ce composant sert à modéliser le compte crédit disponible sur une carte. Le client peut recharger sa carte en ligne. Pour ce faire, le système fait appelle à une API **Bank**. Si le solde est insuffisant lors d'un achat le client doit utiliser un autre moyen de paiement.

PointManager : Ce composant sert ici à modéliser la somme des points disponibles sur le compte du client. Ces même points peuvent être utilisés pour bénéficier des avantages proposés par le système.

PurshaseManager : Ce composant sert à noter les achats de chaque client pour des besoins de statistiques.



Scénarios MVP

Création d'un partenaire et d'un de ses magasins associés par la mairie

- CLI → MayorActionsController → MayorActions → PartnerRepository
- CLI → MayorActionsController → MayorActions → ShopRepository

Le magasin modifie ses horaires d'ouvertures et ajoute une offre avantage simple, non VFP exclusive

- CLI → PartnerActionsController→ PartnerActions → ShopRepository
- CLI → PartnerActionsController→ PartnerActions → AvantageManager → AdvantageRepository

Un consommateur s'inscrit et se connecte

- CLI → ConsumerActionsController → ConsumerActions → ConsumerRepository

Le consommateur regarde les boutiques disponibles

- CLI → ShowCaseController → ShowCase → ShopRepository

Le consommateur fait un achat à la boutique et voit son nombre de points augmenter et l'historique d'achat est crée

- CLI → CardServiceController →CardService -> PointManagerController → PointManager → CardRepository
- CLI → CardServiceController →CardService -> PurchaseManagerController → PurchaseManager → PurchaseRepository

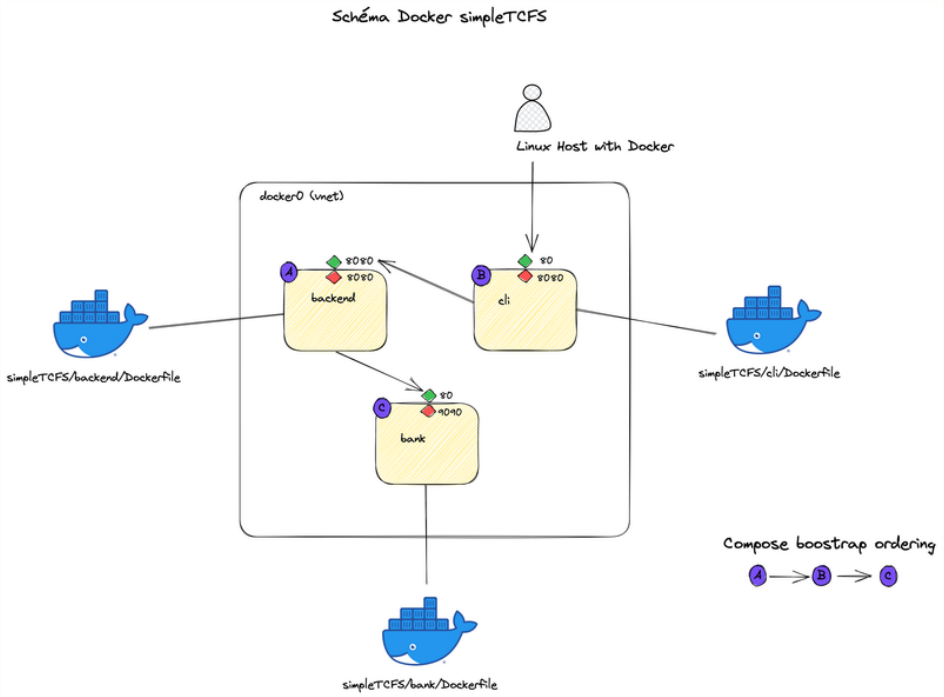
Le consommateur regarde les avantages auxquels il a le droit et en sélectionne un

- CLI → ShowCaseController → ShowCase → AdvantageRepository
- CLI → ConsumerActionsController → ConsumerActions → AvantageManager → SelectedAdvantageRepository

Le consommateur va chercher le cadeau assiocié a l'avantage en magasin et ainsi le vendeur du magasin indique que le cadeau a bien été récupérer

- CLI → CounterCollectController → CounterCollect → UsedAdvantageRepository

Docker chart



On a trois services différents à faire tourner : le backend de l'application, la ligne de commande et la banque. L'utilisateur interagit avec l'application via l'outil en ligne de commande, la cli. La cli récupère les demandes de l'utilisateur et exécute le code correspondant dans l'application. L'application interagit avec la banque pour toutes les actions qui inclut un paiement.

Ports :

- Le conteneur backend exécute le serveur de l'application et est exposé sur le port 8080. Ce qu'il signifie qu'il tourne et écoute les connections TCP sur le même port.
- Le conteneur de la cli tourne sur le port 8080 et écoute les connections TCP sur le port 80. Comme le Dockerfile n'expose pas de port, Docker affecte des ports et un protocole (TCP) par défaut.
- Le conteneur de la banque tourne sur le port 9090 et écoute les connections TCP sur le port 80.

Bootstrap : Le backend qui représente le serveur de l'application doit être démarré en premier. Ensuite, on peut démarrer dans n'importe quel ordre la cli et le service de la banque, peu importe. À savoir que sans la banque, la cli est utilisable mais toutes les actions qui implique un paiement vers la banque renverront une erreur.