POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Rapport d'architecture ISA

Carte multi-fidélités

Équipe F

BAILET Ludovic CHABANIER Aurélia GAZZEH Sourour
HERRMANN Matis YAHIAOUI Imène

Rapport d'architecture ISA

1. Introduction

Dans le cadre de ce cours, nous avons été chargés de créer un logiciel de cartes multi-fidélités pour les villes partenaires afin d'aider les petits commerçants à lutter contre les centres commerciaux. Le présent rapport a pour objectif de vous présenter notre conception de ce logiciel.

1.1. Présentation du sujet

Les cartes multi-fidélités permettent aux adhérents d'obtenir des cadeaux dans les magasins contre des points de fidélité et de profiter de plusieurs avantages dans la ville, comme des places de parking gratuites pendant un certain temps, ou des tickets de bus offerts. Les clients ayant souscrit à ce programme pourront devenir des Very Faithful Person (*VFP*), et obtenir de meilleurs avantages.

1.2. Liste des hypothèses par rapport au sujet

Suite à la lecture du sujet, nous avons émis les hypothèses suivantes :

- La demande de création de compte pour les Commerçants est faite en dehors du système par une demande aux Administrateurs de la zone.
- En tant qu'application municipale, on suppose que toutes les places de parkings de la zone sont concernées par le système multi-fidélités.
- L'envoi des promotions se fait par mail.
- Les commerçants et administrateurs peuvent changer les horaires des magasins en ligne (sur notre application).
- Un utilisateur n'ayant pas de compte peut uniquement souscrire au programme de cartes multi-fidélités.
- Le statut de VFP est renouvelé chaque semaine.
- Pour devenir VFP, il faut avoir utilisé au moins n fois la carte de multi-fidélités avant la fin de la semaine.
- Si un client VFP n'a pas fait ses n achats, il redevient client normal la semaine suivante.
- Les cartes ne se renouvellent pas automatiquement au bout des 18 mois.
- Quand la carte expire, le compte est archivé.

2. Cas d’utilisation

2.1. Acteurs primaires

L’administrateur : Cet acteur va choisir les différentes notifications à envoyer (utilisations restantes pour devenir VFP le samedi, offres promotionnelles, changement d’horaires des magasins). Il va pouvoir aussi créer des comptes pour les commerçants.

Les commerçants : Ces acteurs vont pouvoir envoyer des mails quand ils changent les horaires de leurs magasins.

Les utilisateurs : Ces acteurs vont pouvoir recharger leur carte de multi-fidélité par internet. Ils ont aussi la possibilité d’activer leurs 30 minutes de parking gratuit qu’ils ont obtenues avec l’acquisition du statut VFP.

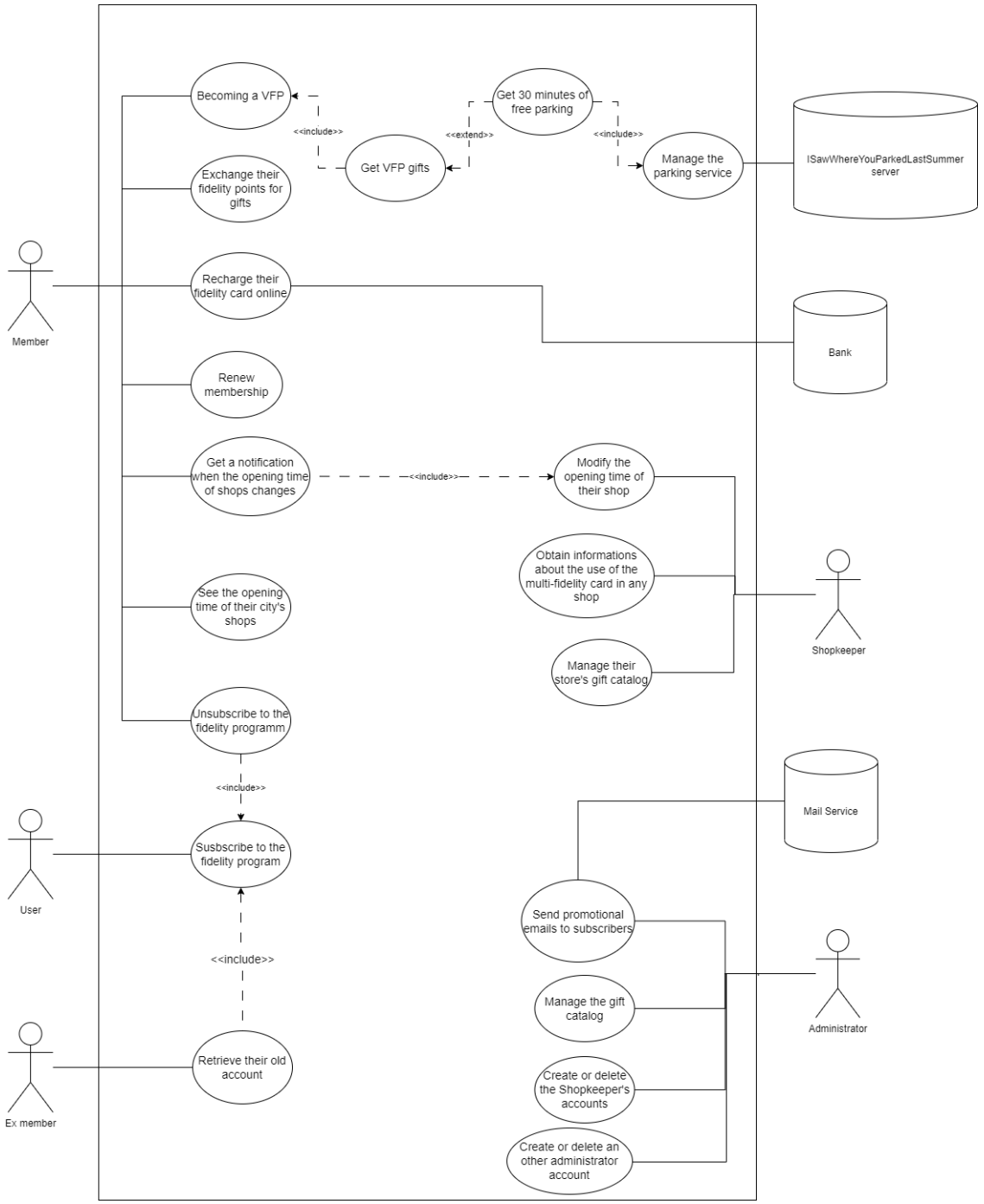
2.2. Acteurs secondaires

La banque : Cet acteur va être notifié lorsqu'un utilisateur va recharger sa carte multi-fidélité.

IsawWhereYouParkedLastSummer : Cet acteur va être notifié lorsqu’un utilisateur va activer ses 30 minutes de parking gratuit.

2.3. Diagramme de cas d'utilisation

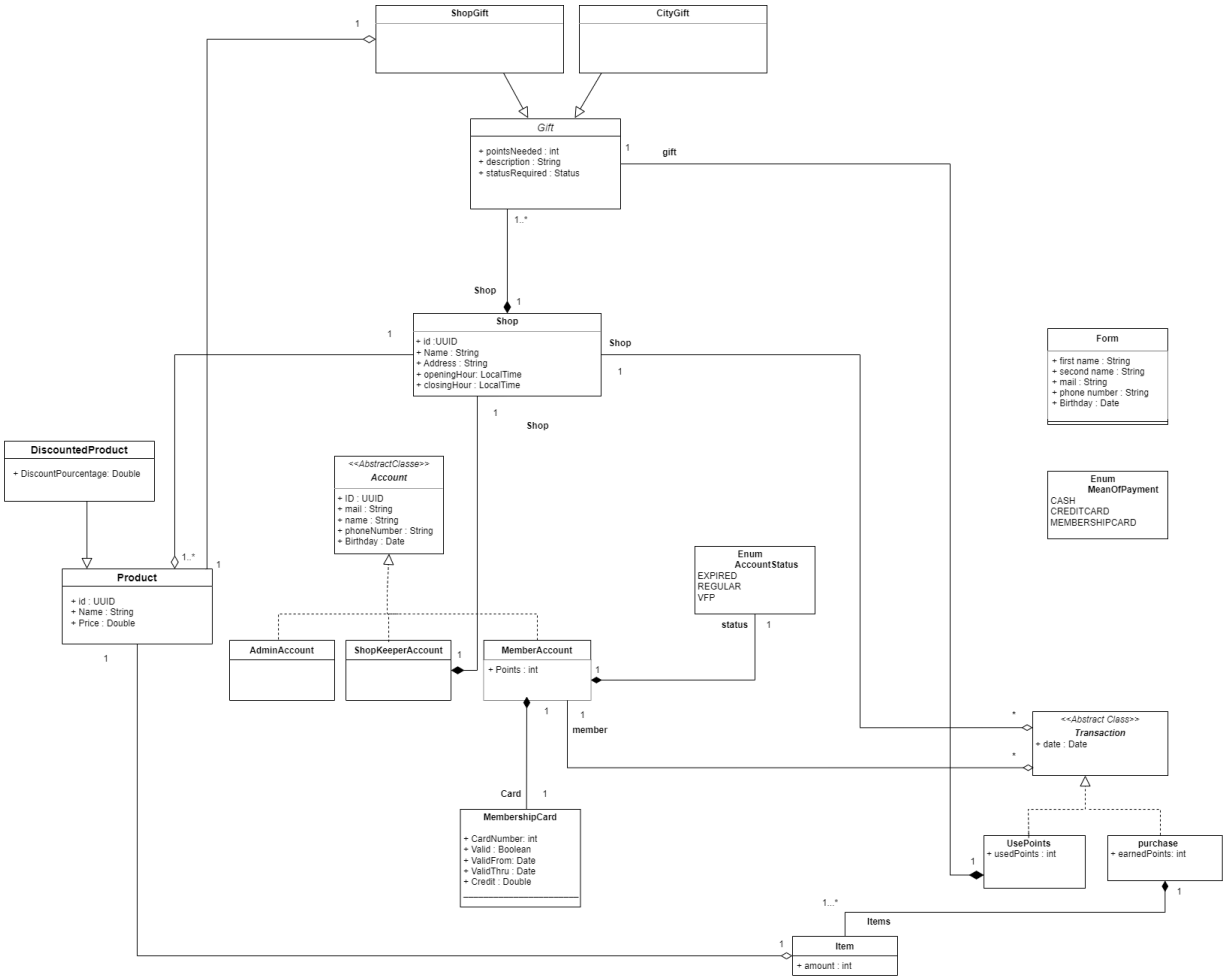
Diagramme de cas d'utilisation



Nous avons choisi de n'avoir que 3 types de comptes : les abonnés (*Member*), les commerçants (*Shopkeeper*) et les administrateurs (*Administrator*). Un utilisateur non abonné ne peut que s'abonner, donc nous avons fait le choix de ne pas créer de compte pour les non-abonnés. De plus, comme dit dans nos hypothèses, les anciens abonnés voient leur compte archivé et peuvent alors le réactiver s'ils se connectent à nouveau.

3. Objets métiers (Diagramme de classes)

Diagramme de classes



La classe **Form** permet de regrouper les informations que le client rentre pour créer ou mettre à jour son compte.

La classe **Account** est abstraite, car elle n'est pas entièrement spécifiée. Pour créer un objet Account, il faut déterminer s'il s'agit d'un compte de commerçant, client ou d'administrateur. Chaque client abonné au service possède un compte (**MemberAccount**). La création d'un compte entraîne automatiquement la création d'une carte associée (**MembershipCard**), et de la même façon, la désactivation d'un compte entraîne la désactivation de la carte associée.

Chaque commerçant possède un seul commerce qui est représenté par la classe **Shop**. Cette classe inclut le nom du commerce, son adresse, les horaires d'ouverture et de fermeture et les produits proposés à la vente **(Product)**.

La classe **DiscountedProduct** hérite de Product et ajoute un attribut **DiscountPourcentage** pour stocker le pourcentage de la remise.

On distingue 2 types de cadeaux pour les clients : les cadeaux qui incluent des produits proposés par le magasin **ShopGift**, et les autres **CityGift** qui incluent des avantages tels que des tickets de bus et des billets pour des attractions.

La classe **Transaction** permet de stocker les informations relatives aux différentes transactions effectuées sur le compte de fidélité, notamment les transactions **Purchase** qui apportent des points et les transactions **UsePoints** qui dépensent les points cumulés.

La classe **Item** permet de regrouper des produits du même type achetés ensembles.

4. Interfaces

```
public interface MemberHandler{

 MemberAccount createAccount(Form form) throws MissingInformationException;
   void archiveAccount(memberAccount memberAccount) throws AccountNotFoundException;
   void restoreAccount(memberAccount memberAccount) throws AccountNotFoundException;
   void deleteAccount(memberAccount memberAccount) throws AccountNotFoundException;
   void updateAccount(memberAccount memberAccount,Form form) throws
AccountNotFoundException;
  void updateAccountsStatus();
  void updateAccountStatus(MemberAccount memberAccount, AccountStatus status) throws
AccountNotFoundException;
}
```

MemberHandler : Cette interface permet d'effectuer tout type d'action nécessaire sur les comptes des membres. Cela inclut leur création, archivage et restauration, mais aussi leur suppression après une certaine période de non-renouvellement de l'abonnement. L'interface a aussi comme rôle de mettre à jour les statuts des comptes (*VFP* ou *Regular*) en fonction du nombre d'achats des membres à chaque période définie (exemple : à la fin de chaque semaine).

```
public interface MemberFinder {
  MemberAccount findById(UUID id);
}
```

MemberFinder : Son rôle est de retrouver un compte membre avec l'identifiant passé en paramètre.

```
public interface ShopRegistry{
  Shop addShop(String name, String address, LocalTime openningHour, LocalTime
closingHour);
 void removeShop(Shop shop);
}
```

ShopRegistry : Cette interface permet à un administrateur d'ajouter ou supprimer un magasin partenaire au programme multi-fidélités.

```
public interface ShopFinder{
  Shop findById(UUID id);}
```

ShopFinder : Son rôle est de récupérer un magasin avec l'identifiant passé en paramètre.

```
public interface ShopKeeperRegistration{
 ShopKeeperAccount createShopKeeperAccount(Form form) throws
missingInformationException;
 void deleteShopKeeperAccount(ShopKeeperAccount account);
}
```

ShopKeeperRegistration: Cette interface permet à un administrateur d'ajouter, de créer ou de supprimer un compte pour un employé qui travaille dans un des magasins partenaires.

```
public interface ShopFinder{
  Shop findById(UUID id);
}
```

ShopFinder : Son rôle est de récupérer un compte de marchand avec l'identifiant passé en paramètre.

```
public interface ShopHandler{
   void setShopHours(Shop shop,LocalTime opening, LocalTime closing) throws
InvalidTimeException;
}
```

ShopHandler : Son rôle est de définir les horaires d'ouverture et de fermeture des magasins.

```
public interface AdminRegistration{
   public AdminAccount createAdminAccount(Form form) throws
missingInformationException;
 public void deleteAdminAccount(AdminAccount  account);}
```

AdminRegistration : Cette interface permet à un administrateur de créer ou supprimer un autre compte administrateur.

```
public interface AdminFinder{
  Shop findById(UUID id);
}
```

AdminFinder : Son rôle est de récupérer un compte administrateur avec l'identifiant passé en paramètre.

```
public interface TransactionProcessor{
  void processPurchase(Purchase purchase);
  void processPointsUsage(UsePoints usePoint);
}
```

TransactionProcessor : Cette interface sert à l'achat de produits dans les magasins partenaires, et permet d'utiliser ses points.

```
public interface Payment{
  void payment(Purchase purchase, MeanOfPayment payment) throws PaymentDenied;  }
```

Payment : Cette interface sert à lancer la procédure du paiement d'un achat.

```
public interface PointTrader{
  void tradePoints(Transaction transaction);
}
```

PointTrader : Le rôle de cette interface est d'attribuer ou enlever des points à un membre à la suite d'une transaction.

```
public interface MailSender{
  void sendPromotions();
  void sendVFPReminders(); }
```

MailSender : Cette interface permet aux administrateurs d'envoyer des mails promotionnels aux membres, ou de rappeler à ceux qui détiennent le statut *VFP* de la possibilité de la perte de leur statut s'ils ne sont pas actifs.

```
public interface TransactionExplorer{
  void getShopTransactions(Shop shop);
}
```

TransactionExplorer : Cette interface permet aux commerçants de consulter les achats effectués et les cadeaux qui ont été offerts dans n'importe quel magasin partenaire.

```
public interface CatalogEditor{
  void editCatalog(List<Product> addedProducts, List<Products> removedProducts);
}
```

CatalogEditor : Cette interface permet aux administrateurs de modifier le catalogue global du programme multi-crédit.

```
public interface ShopCatalogEditor{
  void editShopCatalog(List<Product> addedProducts, List<Products> removedProducts);
}
```

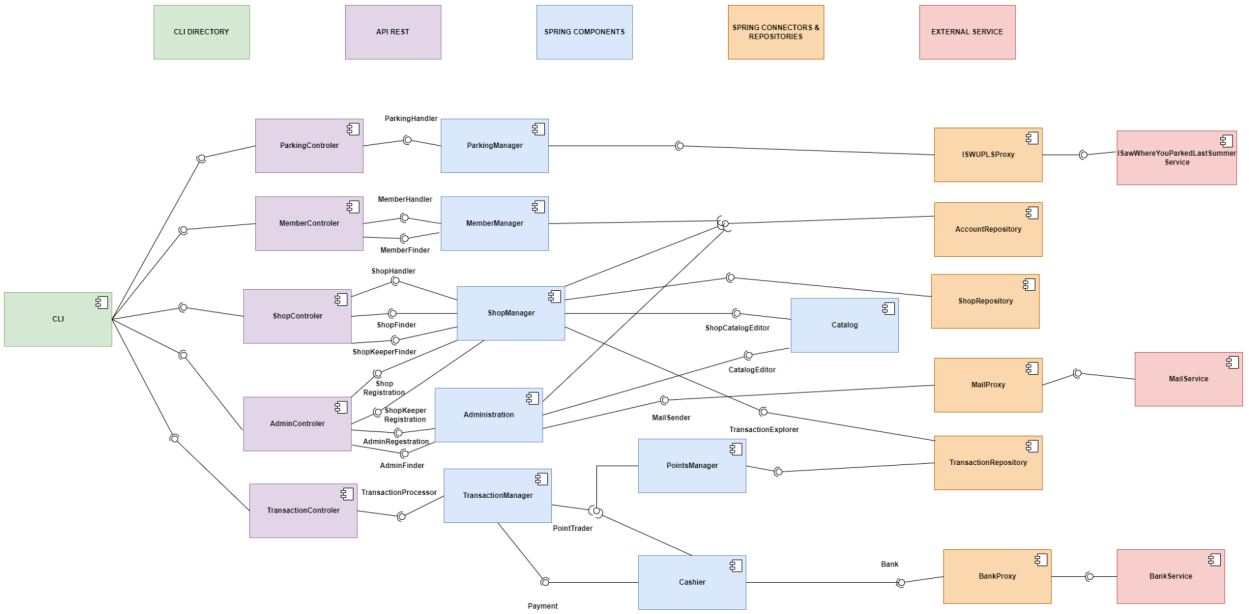
ShopCatalogEditor : Cette interface permet aux marchands de modifier la section correspondant à leurs magasins dans le catalogue.

```
public interface ParkingHandler{
  void registerParking(MemberAccount account, String CarRegistrationNumber) throws
NotVFPException;
}
```

ParkingHandler : Cette interface permet aux membres *VFP* de profiter de leur temps de parking offert en communiquant leur numéro d'immatriculation à la plateforme **IsawWhereYouParkedLastSummer.**

5. Composants

Diagramme de composants



MemberManager : Ce composant gère tout ce qui est création, mise à jour, archivage et suppression des comptes des membres. Il permet aussi de restaurer un compte archivé à la réinscription d'un ex-membre et de retrouver un compte à partir de son identifiant.

ShopManager : Ce composant s'occupe de l'enregistrement des magasins partenaires, de la création des comptes des commerçants et de la modification des horaires d'ouverture. Il permet aussi de retrouver les magasins et les comptes des marchands depuis leurs identifiants.

Administration : Ce composant gère les tâches administratives à l'exception de l'enregistrement ou suppression des magasins et des comptes marchands. Il s'occupe de la création et suppression des comptes administrateurs, ainsi que de l'envoi des mails de promotions. Il permet aussi de retrouver un compte administrateur depuis son identifiant.

TransactionManager : Ce composant gère le traitement des transactions des membres. Cela regroupe les achats et l'échange de points contre des récompenses.

Cashier : Ce composant s'occupe du lancement et de l'enregistrement des paiements.

PointsManager : Ce composant gère l'attribution ou la soustraction des points lors des achats et des attributions de cadeaux.

Catalog : S’occupe de la gestion et de la mise à jour du catalogue. Il permet aux administrateurs de modifier le catalogue global et au commerçants de modifier la section de leurs magasins.

ParkingManager : Ce composant gère l’utilisation des 30 minutes de parking par les membres avec le statut VFP. Il permet l’envoie du numéro d'immatriculation à l'application **IKnowWhereYouParkedLastSummer** via son proxy.

6. Scénarios MVP

Nous envisageons un ensemble de scénarios pour construire un MVP et nous l'étendrons ensuite pour couvrir l'ensemble des fonctionnalités du projet.

Ce MVP consiste en :

- L'inscription d'un commerçant au service
- L'inscription d'un client
- Le rechargement de la carte en ligne
- L'utilisation de la carte uniquement dans la ville renseignée lors de l'inscription
- L'échange de points contre des cadeaux ou des titres de transports
- Le gain de points lors d'achats dans les magasins partenaires.

7. Docker Chart de la Cookie Factory fournie (DevOps)

[Docker Chart de la CookieFactory fournie (DevOps)]