ISA/Devops

Carte multi-fidélités

Équipe H

(comme hibou)

Buquet Antoine

-

Karrakchou Mourad

-

Imami Ayoub

-

Bonnet Kilian

-

Le Bihan Léo

I. Introduction

Rappel rapide du sujet

Le principe des cartes multi-fidélités est de fidéliser un client sur une zone commerciale donnée, ainsi en jouant collectif, tous les commerçants de la zone y gagnent. Ces systèmes sont déployés en collaboration avec les mairies qui sponsorisent ce type d'action via un catalogue cadeau disponible pour les abonnés à la carte multi-fidélités, comme des réductions sur le parking ou les transports en commun par exemple.

Nous souhaitons donc développer non pas un système pour une ville donnée, mais une architecture logicielle que l'on pourrait réutiliser dans différentes villes. Ce système de carte multi-fidélités doit permettre aux clients :

- de gagner des points proportionnellement aux montants dépensés chez les partenaires.
- de charger une faible somme d'argent sur la carte pour des petits achats chez les partenaires, via un chargement en ligne par carte bleue.
- de débloquer certains avantages institutionnels, comme un ticket de bus offert par jour, tant qu'ils gardent le statut *Very Faithful Person* - VFP, obtenable à partir d'une fréquence d'utilisation hebdomadaire de la carte.

Liste des hypothèses par rapport au sujet

Zones

- L'application est conçue pour une localité.
- Les commerces partenaires peuvent être couverts par une seule zone.
- Une carte multi-fidélité est utilisable dans une zone géographique précise.
- Une carte multi-fidélité ne peut être utilisée uniquement dans les commerces participants au programme de cette zone.

Soldes

- La carte multi-fidélité possède deux soldes : un solde de points de fidélité et un solde de liquidité (€).
- Le solde maximum dans le compte fidélité ne peut excéder 200€.
- Le solde de liquidité (€) après achat ne peut pas être inférieur à 0. Il n'y a pas de crédit.
- Le solde de points de fidélité après dépense des points ne peut pas être inférieur à 0.

Points fidélité

- Un commerçant peut échanger un bien ou un service contre un certain nombre de points de fidélité.
- Le solde de points de fidélité ne peut pas être inférieur à 0 après un achat.
- Le client effectuant un achat dans un commerce partenaire et ayant sa carte multi-fidélité obtient une récompense en point de fidélité proportionnel au montant de ses achats.

Statut VFP

- À chaque début de mois, la fréquence d'utilisation hebdomadaire de chaque client est calculée afin de déterminer qui peut obtenir le statut de Very Faithful Person-VFP.
- Une personne ayant acquis le statut VFP peut perdre ce statut le mois suivant, si sa fréquence d'achat se réduit.
- Un client ayant le statut VFP obtient des avantages institutionnels.
    - 1 ticket aller-retour de bus offert par jour.
    - 30 minutes de parking gratuit par jour.

II. Cas d'utilisation

Acteurs primaire

- UserNotRegister
- Client : c'est l'utilisateur principal de l'application, il effectue ses achats dans les magasins associés au programme pour obtenir des points sur sa carte de multi-fidélité.
- Administrateur : il travaille à la mairie, c'est lui qui doit enregistrer les comptes des magasins participant au programme.
- Commerçant partenaire : il propose des avantages et des réductions pour le client disposant de la carte multi-fidélité.
- Collectivité territoriale associée : elle permet d'offrir certains avantages comme un ticket de bus offert par exemple.

Légende:

- **Acteur en gras**
- *Cas d'utilisation en italique*

Les personnas :

Jacques est un **UserNotRegister** qui va *s'enregistrer à notre programme de carte de fidélité*. Il devient alors un **Client** et la régularité de ses *achats* va lui permettre d'*obtenir le statut de VFP*. Il sera donc *authentifié sur un compte VFP* et pourra alors *visualiser le catalogue de cadeau et d'avantage* puis *utiliser un avantage* (le ticket de bus gratuit).

Alison est une **cliente** qui possède la carte multi-fidélité. Elle aime utiliser ses points en échange de cadeau, auprès des commerçants partenaires. Obtenir un cadeau nécessite l'achat d'un produit chez le commerçant, elle va donc *faire un achat* auprès de son commerçant préféré, ce qui va lui permettre de *gagner des points* mais aussi d'être éligible au fait d'*obtenir un cadeau.*

Franck travaille à la mairie, il est **administrateur système**. De ce, il représente la mairie et la collectivité territoriale au sein du système. La collectivité territoriale souhaite offrir des places de bus aux utilisateurs VFP, Franck va être chargé de *modifier le catalogue d'avantages* pour ajouter les places de bus. De plus, la mairie a reçu plusieurs demandes d'ajout de commerces partenaires, Franck va donc *créer les comptes des commerçants dans le système*. La prochaine tâche qu'il lui a été assigné est d'*envoyer des offres promotionnelles*, des *sondages de satisfaction* et *relancer les consommateurs qui vont*

perdre leurs statuts de VFP. Enfin pour finir sa journée Franck va r*écupérer des indicateurs chiffré* afin de les faire remonter à la mairie.

Pierre est un **client,** la régularité de ses *achats* va lui permettre d'*obtenir le statut de VFP*. Il va *modifier son profil* pour ajouter sa plaque d'immatriculation. Il peut ensuite *utiliser un avantage* pour avoir ses 30 minutes de parking gratuit.

Laura est **une commerçante partenaire.** Elle peut obtenir *des indicateurs chiffrés* pour avoir les informations nécessaires afin de convaincre ses patrons et de *modifier le catalogue des cadeaux* pour pouvoir changer l'avantage disponible*.*

Acteurs secondaire

- ISawWhereYouParkedLastSummer : plateforme externe utilisée par toutes les villes où le programme est mis en place. Il permet de gérer les avantages VFP liés aux places de parking.
- Banque : plateforme externe utilisée afin de recharger la carte multi-fidélité du client avec sa carte bancaire.

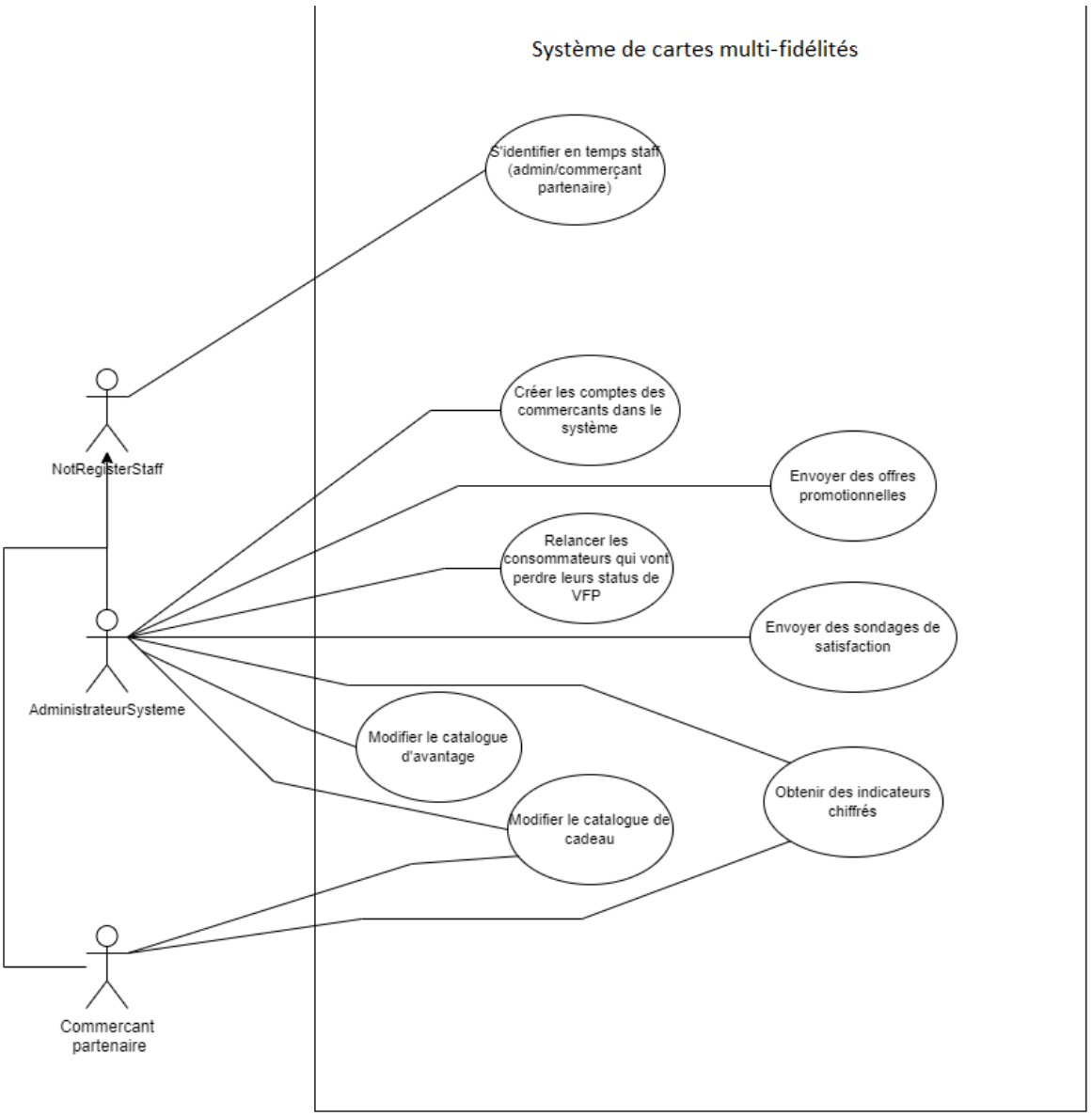
Les personnas :

Le **Service de Banque** permet lorsqu'un client souhaite *recharger sa carte multi-fidélité*, de *débiter de l'argent* de la carte bancaire du client.

Le service **IsawWhereYouParkedLastSummer** fournis en partenariat avec la collectivité territoriale 30min de parking gratuit. Lorsqu'un client souhaite *consommer cet avantage*, le service **IsawWhereYouParkedLastSummer** *offre 30 minutes de stationnement* au client.

Diagramme des cas d'utilisations

Le diagramme étant très grand, nous avons décidé de le découper en deux afin que nous puissions l'afficher sur deux pages et ainsi rendre sa lecture plus confortable.



Pour découper ce sujet afin d'en faire un diagramme de cas d'utilisation, nous avons procédé en suivant les étapes suivantes :

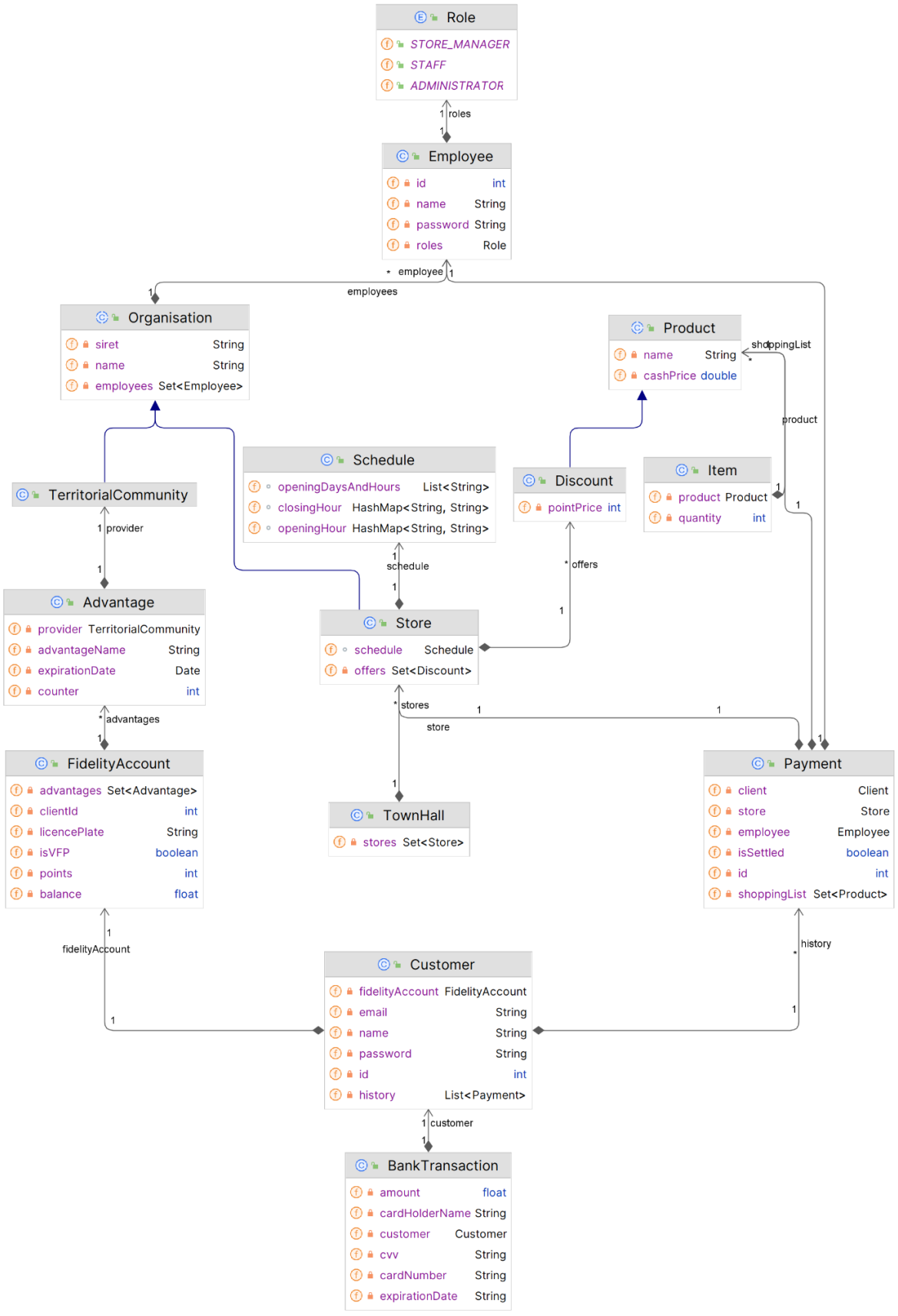
1. Identifier les acteurs: Il y a plusieurs acteurs clés dans ce système, tels que les clients, les policiers, les commerçants participants, les administrateurs du système et les collectivités territoriales.

2. Définir les cas d'utilisation : En utilisant les informations fournies dans le sujet, nous pouvons définir les cas d'utilisation suivants :

- Inscription au système : Le client s'inscrit au système en fournissant des informations personnelles telles que son nom, son adresse e-mail et son mot de passe.
- Ajout de points de fidélité : Le client gagne des points de fidélité en achetant chez le commerçant participant.

- Utilisation des points de fidélité : Le client utilise ses points de fidélité pour obtenir des récompenses telles que des réductions ou des avantages supplémentaires.
- Chargement de fonds sur la carte : Le client charge de l'argent sur sa carte pour effectuer des achats chez le commerçant participant.
- Obtention du statut VFP : Le client atteint un certain niveau d'utilisation de la carte et obtient le statut VFP, lui donnant accès à des avantages supplémentaires.
- Gestion des comptes : L'administrateur du système gère le compte du client et du commerçant participant, y compris la modification des informations de compte et la suppression de comptes inactifs.
- Ajout de commerçant participant : Le commerçant peut s'inscrire pour devenir un commerçant participant et s'afficher en tant que tel au client.

III. Objets métiers



Customer : L'objet customer à pour objectif de centraliser les données correspondant à un client. Un customer se définit essentiellement par son id et de son mot de passe qui lui serviront à s'authentifier sur notre application. De plus, pour pouvoir profiter des avantages offerts par sa carte de fidélité, un client possède également un objet FidelityAccount. Afin de garder une trace des paiements effectués, un client possède également une liste de ses paiements qui représentent ici son historique d'achat.

FidelityAccount : Détenu par un client, la classe représente le compte fidélité et centralise essentiellement les différents compteurs nécessaires au suivi de la fidélité d'un client.

- Compteur "points" : Représente le nombre de points de fidélité que possède un client.
- Compteur "balance" : Représente le solde que possède un client sur sa carte.
- Booléen "isVFP" : Indique si le client à le statut VFP ou non.
- La liste d'avantages appelée "avantages" permet de suivre la consommation d'un avantage.

Avantage : Un avantage représente les différents avantages institutionnels que peuvent obtenir les clients ayant le statut VFP (1 ticket de bus offert par jour, par exemple). Cet objet sert essentiellement de compteur afin de suivre la consommation d'un avantage par un client avant la réinitialisation de cet avantage.

- ExpirationDate : Représente la date à laquelle le compteur de l'avantage se réinitialise.
- Provider : Représente l'organisation offrant cet avantage. Envibus serait le provider de l'avantage "1 ticket de bus offert par jour" par exemple.
- Counter : Le compteur permet de suivre la consommation de l'avantage par le client. Ce compteur se décrémente de 1 à chaque consommation de cet avantage et ce réinitialise lorsque la date d'expiration est atteinte.

Organisation : Une organisation représente une infrastructure gérant du personnel en général. Une organisation est composée de son nom, de son identifiant SIRET et possède également une liste d'employés.

Employee : Cet objet centralise les données correspondant à l'employé. Un employé se caractérise essentiellement par son nom, son numéro d'identification ainsi que de son mot de passe permettant de valider son identité pour interagir avec le système. L'employé possède également un rôle permettant de lui accorder ou non différentes permissions d'interaction avec le système.

Role : Cette classe énumérée représente les 3 différents rôles que peut posséder un employé lorsque ce dernier interagit avec notre système.

- ADMINISTRATOR : Peut -entre-autre- ajouter répertorier un nouveau magasin partenaire et ajouter de nouvelles offres proposées par un magasin.
- STAFF : Peut modifier les horaires d'ouvertures d'un magasin.
- STAFF : Peut enregistrer les transactions faites par un client sur notre système.

Store : Store est la représentation métier du magasin. Un magasin hérite d'une organisation : Il possède ainsi un nom, un numéro d'authentification ainsi que d'une liste d'employés. Un magasin possède cependant des indications sur ses horaires d'ouverture à l'aide de l'objet Schedule ainsi qu'une liste de Discount que propose le magasin.

Produit : Un produit est représenté par son nom et son prix.

Item : Un article est une classe d'association permettant de lier un produit à une quantité.

Discount : Hérité de produit, un Discount représente une offre que peut proposer un client. Il possède en plus d'un nom et d'un prix un coût en points. Dans le cas d'un café gratuit à 2 points, nous pouvons imaginer que les champs ont été rempli de la manière suivante :

- Name : "Café gourmand";
- cashPrice = 3; // Le café coûtait originellement 3€
- pointPrice = 2; // Le café peut également être acheté pour 2 points;

Il est important dans notre implémentation de garder le prix initial du produit afin de pouvoir faire remonter ces données sous forme de statistiques. Par exemple, une statistique montrant quel pete fait un magasin en proposant le café gratuit à ce prix là.

Payment : L'objet Payment représente la forme sous laquelle un achat est traité par notre système. Un achat se caractérise par :

- Customer : Le client ayant effectué l'achat
- Store : Le magasin où l'achat a été effectué
- Employee : L'employé ayant procédé à la vente
- isSettled : Un booléen permettant de savoir si l'achat a déjà été réglé ou si le montant doit être ponctionné sur le solde de la carte fidélité du client.
- shoppingList : Une liste d'articles permettant de connaître le montant total en point à ponctionner et le montant total en cash à convertir en points et/ou à débiter du solde de la carte de fidélité du client.

BankTransaction : Représente l'objet utilisé lors de la communication avec le composant chargé de discuter avec la banque. Il centralise toutes les données nécessaires pour effectuer et suivre une transaction bancaire. Une transaction bancaire se caractérise par :

- Customer : Le client souhaitant recharger son solde
- Amount : Le montant à prélever sur son compte en banque et à créditer sur le solde de sa carte de fidélité
- cardHolderName : Le nom du détenteur de la carte bancaire
- cardNumer : Le numéro de carte
- cvv : Le cvv de la carte
- expirationDate : La date d'expiration de la carte

IV. Interfaces

- CustomerExplorer : Permet de vérifier l'identité d'un client et de vérifier l'intégrité d'une connexion.

```
public interface CustomerExplorer {
   void checkCredentials(String name, String password) throws
BadCredentialsException, MalformedCredentialsExceptions;
}
```

- CustomerRegistration : Inscrire un nouveau client au système pour lui permettre de se connecter.

```
public interface CustomerRegistration {
   Customer register(String name, String mail, String password) throws
MailAlreadyUsedException;
}
```

- CustomerModifier : Permet de mettre à jour les données d'un client.

```
public interface CustomerModifier {
   void modifyUsername(Customer customer);
   void modifyMail(Customer customer) throws MailAlreadyUsedException;
   void modifyPassword(Customer customer);
}
```

- BalanceModifier : Permet d'incrémenter et de décrémenter le solde d'un client.

```
public interface BalanceModifier {
    void decreaseBalance(Customer customer, float amount) throws
NotEnoughBalanceException;
    void rechargeBalance(Customer customer, BankTransaction bankTransaction,
float amount) throws MalformedBankInformationException;
}
```

- PointModifier : Permet d'incrémenter et de décrémenter le nombre de points de fidélité d'un client.

```
public interface PointModifier {
    //points computed from the price
    void incrementPoints(Customer customer, float price);
    void decrementPoints(Customer customer, int points);
}
```

- IPayment : Permet de faire un achat.

```
public interface IPayment {
   void pay(Payment payment, String EmployeeName, String employeePassword)
throws NotEnoughBalanceException, WrongEmployeeNameOrPassword;
}
```

- FidelityCardPurchase : Permet de faire un achat en utilisant l'argent préalablement déposé sur sa carte multi-fidélité.

```
public interface FidelityCardPurchase {
   void buyWithFidelityCard(Customer customer, Payment payment, Store store)
throws NotEnoughBalanceException;
}
```

- PointPurchase : Permet de faire un achat en utilisant les points accumulés sur sa carte multi-fidélité.

```
public interface PointPurchase {
   void buyWithPoint(Customer customer, Payment payment, Store store) throws
NotEnoughBalanceException;
}
```

- SettledPurchase : Permet une fois l'achat vérifié et validé, d'incrémenter le nombre de points sur la carte de fidélité du customer.

```
public interface SettledPurchase {
   void validatePurchase(Customer customer, Payment payment, Store store);
}
```

- PaymentModifier : Permet d'ajouter un paiement dans le PaymentRepository.

```
public interface PaymentModifier {
   void savePayment(Payment payment) throws PaymentAlreadyExistsException;
}
```

- RefillFidelityCard : Permet de mettre de l'argent sur sa carte multi-fidélité.

```
public interface RefillFidelityCard {
   void refill(Customer customer, BankTransaction transaction, float amount)
throws MalformedBankInformationException;
}
```

- FidelityExplorer : Permet d'obtenir un compte de fidélité en fonction d'un client.

```
public interface FidelityExplorer {
   FidelityAccount getFidelityAccountByCustomer(Customer customer) throws
CustomerNotFoundException, FidelityAccountNotFoundException;
List<FidelityExplorer> getAllEligibleVFPCustomer();
}
```

- PaymentExplorer : Permet d'obtenir le paiement à partir d'un client ou d'un magasin.

```
public interface PaymentExplorer {
   Payment getPaymentByCustomer(Customer customer) throws
CustomerNotFoundException, NoPaymentFoundException;
   Payment getPaymentByStore(Store store) throws StoreNotFoundException,
NoPaymentFoundException;
}
```

- AdvantageConsumer : Cette interface permet au client de consommer les avantages qu'il a à sa disposition.

```
public interface AdvantageConsumer {
   void consumeAdvantage(Advantage advantage) throws NoAdvantageFoundException;
}
```

- CustomerAdvantageExplorer : Cette interface permet au client de visualiser les avantages qu'il possède.

```
public interface CustomerAdvantageExplorer {
   List<Advantage> getAdvantage(int userId) throws  NoAdvantageFoundException;
}
```

- VFPTransaction: Cette interface permet au client de réaliser la transaction qui va lui permettre d'utiliser un ses avantages.

```
public interface VFPTransaction {
   boolean tryUseAdvantage(String username, Advantage advantage) throws
CustomerNotFoundException, NotVFPClientException,
ISawWhereYouParkedLastSummerUnvailableException;
}
```

- EmployeeRegistration : Permet à un employé d'ajouter un nouvel employé dans les registres

```
public interface EmployeeRegistration {
    void addEmployee(Employee employee, String newEmployeeName, String
newEmployeePassword, Role newEmployeeRole) throws NotEnoughPermissionException,
EmployeeNotFoundException;
}
```

- EmployeeFinder : Permet de vérifier l'identité d'un administrateur ou commerçant partenaire

```
public interface EmployeeFinder {
   Employee findEmployeeById(int id) throws EmployeeNotFoundException;
   Set<Employee> findEmployeeByName(String name);
   void checkCredentials(String name, String password) throws
BadCredentialsException, MalformedCredentialsExceptions;
}
```

- AdvertiserManager : Cette interface permet à l'administrateur de créer des offres promotionnelles, des sondages de satisfactions et de relancer les consommateurs, par exemple lors de la perte de leur statut de VFP.

```
public interface AdvertiserManager {
    void createPromotionalOffer(int userId) throws CustomerNotFoundException,
NotEnoughPermissionException;
    void createSatisfactionSurvey(String surveyForm);
    void createReminderMessage(int userId) throws CustomerNotFoundException,
NotEnoughPermissionException;
}
```

- StoreModifier : Cette interface permet aux employés de modifier les informations et les employés du store.

```
public interface StoreModifier {
   void addEmployee(String employeeName, String employeePassword, String
newEmployeeName, String newEmployeePassword, Role newEmployeeRole) throws
NotEnoughPermissionException, EmployeeNotFoundException,
WrongEmployeeNameOrPassword;
   void deleteEmployee(int id, String myName, String myPassword) throws
NotEnoughPermissionException, EmployeeNotFoundException,
WrongEmployeeNameOrPassword;
   void changeDayOpeningHours(String Day,String openingHour,String closingHour,
String myName, String myPassword) throws NotEnoughPermissionException,
WrongEmployeeNameOrPassword, InvalidDayException, InvalidHourException;
   void changeDayStatus(String Day,Boolean open, String myName, String
myPassword) throws NotEnoughPermissionException, WrongEmployeeNameOrPassword,
InvalidDayException, InvalidHourException;
}
```

- StoreFinder : Permet de rechercher un store par id et de vérifier l'intégrité de la demande.

```
public interface StoreFinder {
   Store findStore(String name, String myName, String myPassword)  throws
BadCredentialsException, StoreNotFoundException;
}
```

- StoreRegistration : Permet à l'admin d'enregistrer un nouveau magasin.

```
public interface StoreRegistration {
   void registerNewStore(String storeName, String storeSiret, String userName,
String password) throws BadCredentialsException, NotEnoughPermissionException,
MissingInformationsException;
}
```

- StoreExplorer : Permet d'avoir des informations relatives au store.

```
public interface StoreExplorer {
   Schedule getOpeningHours(Store store) throws StoreNotFoundException;
}
```

- ScheduleExplorer : Permet d'obtenir les horaires d'ouverture et de fermeture d'un magasin à un jour donné.

```
public interface ScheduleExplorer {
   boolean isStoreOpen(Store store) throws StoreNotFoundException;
   String getOpeningHour(Store store, String day) throws StoreNotFoundException,
InvalidDayException;
   String getClosingHour(Store store, String day) throws StoreNotFoundException,
InvalidDayException;
}
```

- StatsExplorer : Fournit des rapports entre le total des avantages/réductions offerts et le total des paiements.

```
public interface StatsExplorer {
   //return the ratio between the total of people payment and the total of the
advantage unlocked
   double advantageRatio();
   //return the ratio between the total of people payment and the total of the
discount unlocked
   double discountRatio();
   //return the ratio between the total of people payment and the total of the
advantage unlocked for a given store
   double advantageRatioByStore(String storeName, String myName, String
myPassword);
   //return the ratio between the total of people payment and the total of the
discount unlocked for a given store
   double discountRatioByStore(String storeName, String myName, String
myPassword);
}
```

- DiscountModifier : Fournit la possibilité de modifier les réductions du système

```
public interface DiscountModifier {
   void modifyPointPrice(Discount discount, int newPointPrice) throws
DiscountNotFoundException;
}
```

- ScheduleModifier : Permet de changer les horaires d'ouvertures d'un magasin

```
public interface ScheduleModifier {
    void changeDayOpeningHours(Schedule schedule, String Day, String
openingHour, String closingHour) throws NotEnoughPermissionException,
WrongEmployeeNameOrPassword, InvalidDayException, InvalidHourException;
    void changeDayStatus(Schedule schedule, String Day,Boolean open) throws
NotEnoughPermissionException, WrongEmployeeNameOrPassword, InvalidDayException,
InvalidHourException;
}
```

- DiscountExplorer : Permet d'obtenir les informations concernant les réduction

```
public interface DiscountExplorer {
   Discount findDiscountByName(Store store, String discountName) throws
DiscountNotFoundException;
   List<Discount> getAllDiscount() throws NotEnoughPermissionException;
}
```

- AdvantageModifier : Fournit la possibilité de modifier les avantage du système

```
public interface AdvantageModifier {
    void modifyAdvantageName(Advantage advantage, String newName) throws
AdvantageNotFoundException;
    void modifyExpirationDate(Advantage advantage, String newDate) throws
AdvantageNotFoundException;
    void modifyCounter(Advantage advantage, int newCounter) throws
AdvantageNotFoundException;
}
```

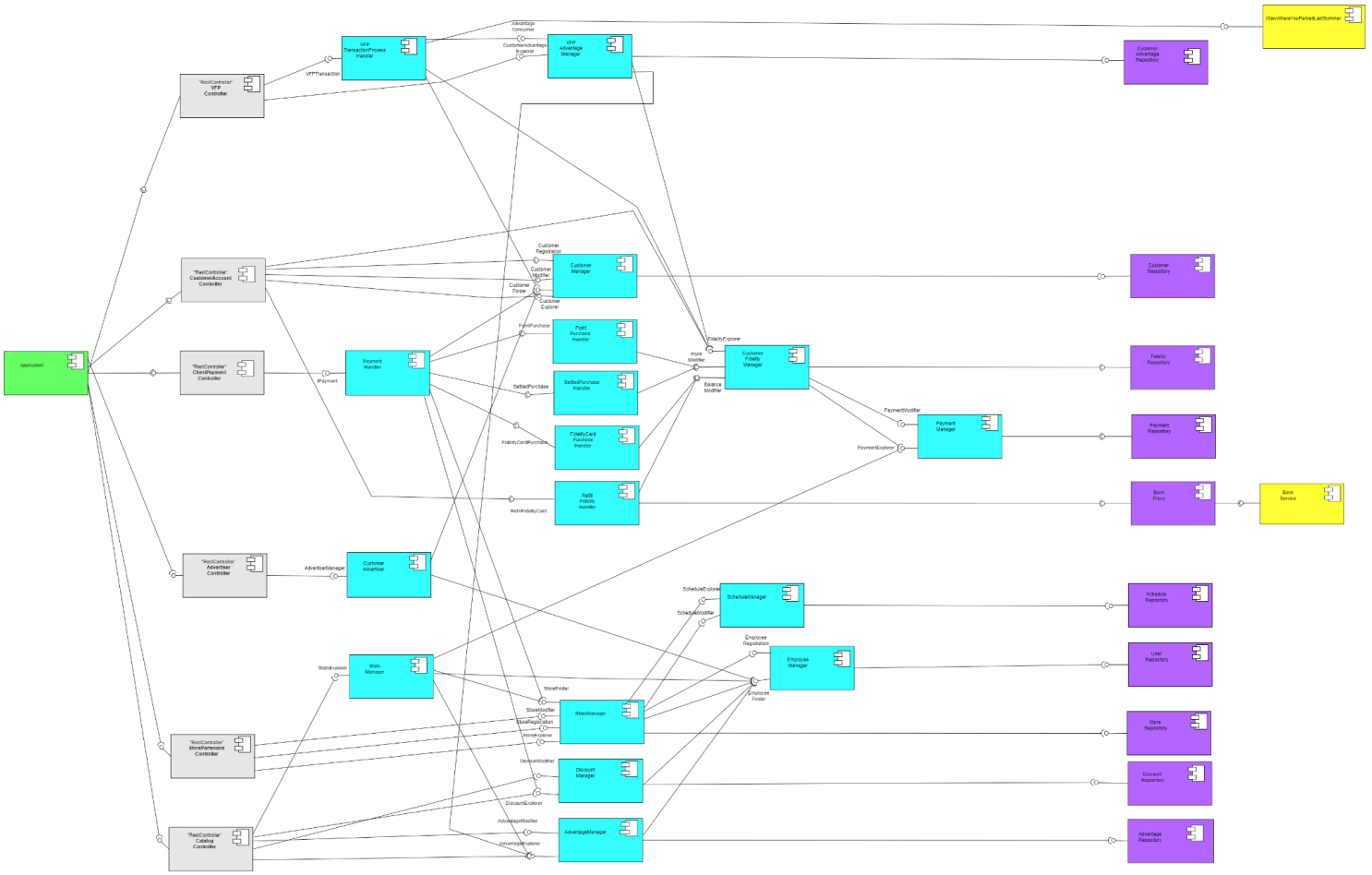
- AdvantageExplorer : Permet d'obtenir les informations concernant les avantages

```
public interface AdvantageExplorer {
   List<Discount> getAllAdvantage() throws NotEnoughPermissionException;
}
```

- CustomerFinder : Permet de trouver un client via l'id ou un set de clients via un nom.

```
public interface CustomerFinder {
    Customer findCustomerById(int id) throws CustomerNotFoundException;
    Set<Customer> findCustomersByName(String name);
}
```

V. Composants



Meilleure qualité ici.

REST

CustomerAccountController (REST API) : Permet d'inscrire un nouveau client ou de modifier les données d'un client.

CustomerFidelityController (REST API) : Permet de consulter les informations concernant la fidélité d'un client passé par id (ex. point de fidélité).

StoreDiscountController (REST API) : Permet d'inscrire, de modifier ou de supprimer une remise en fonction de l'id d'un magasin donné.

SPRING

Composant VFPAdvantageManager : Le composant VFPAdvantageManager gère les avantages du client (Customer) VFP. Il reçoit des demandes de VFPTransactionHanddler pour consommer les avantages que le client possède (par l'interface AdvantageConsumer). De plus, il expose les avantages du client auprès du VFPController (par l'interface AdvantageExplorer) en interrogeant le CustomerAdvantageRepository. Enfin, le composant

VFPAdvantageManager demande au composant AdvantageManager les différents avantages proposés par la mairie/collectivité territoriale en utilisant l'interface AdvantageExplorer. De plus, au début de chaque mois il utilise l'interface FidelityExplorer pour trouver tous les clients éligibles au statut VFP et utilise l'interface AdvantageExplorer pour trouver quels Advantages donner aux clients.

VFPTransactionProcessHandler : Le composant VFPTransactionProcessHandler à pour rôle de gérer le processus des transactions qui concernent l'utilisation des avantages. Il expose au VFPController à travers l'interface VFPTransaction, le moyen d'utiliser les avantages que possède le client. Pour ça, il va demander au composant CustomerManager l'identité du client à travers l'interface CustomerFinder, il va vérifier que le client a bien le grade VFP en interrogeant le composant CustomerFidelityManager à travers l'interface FidelityExplorer. Enfin une fois que le composant à identifier le client et vérifié qu'il possède bien le grade VFP, alors il va demander au composant VFPAdvantageManager de consommer l'avantage désiré.

CustomerAdvertiser : Le composant CustomerAdvertiser à pour rôle créer des offres promotionnelles, créer des sondages de satisfactions, relancer les consommateurs, par exemple lors de la perte de leur statut de VFP (par l'interface AdvertiserManager). Cependant seul les administrateurs peuvent accéder à ces fonctionnalités, le component va demander au composant EmployeeManager si l'utilisateur est autorisé à réaliser ces actions (par l'interface EmployeeFinder)

PaymentHandler : Le composant PaymentHandler va gérer la manière dont le paiement du client va être traité via les différentes interfaces qu'il implémente (par l'interface Payment). Il va vérifier l'identité du client en interrogeant le composant CustomerManager (à travers l'interface CustomerFinder). Afin d'utiliser les points du client, le composant va demander au composant PointPurchaseHandler (à travers l'interface PointPurchase). De plus, pour procéder à l''utilisation du solde de la carte multi-fidélité, le composant va s'adresser au composant FidelityCardPurchaseHandler (via l'interface FidelityCardPurchase). Le composant va demander à identifier le store qui réalise la transaction en demandant au composant StoreManager (via l'interface StoreFinder). Enfin il va demander les discount disponibles à DiscountManager (par l'interface DiscountExplorer).

PointPurchaseHandler : Le composant PointPurchaseHandler a pour rôle de traiter le paiement d'un client qui achèterait un article avec les points qu'il dispose sur sa carte multi-fidélité.

SettledPurchaseHandler : Le composant gère les achats vérifiés et validés. Il reçoit en entrée le Customer qui a effectué l'achat, le Store ou a été effectué l'achat et la description du paiement. Il utilise l'interface PointModifier afin d'incrémenter nombre de points sur la carte du client proportionnellement au montant de son achat.

FidelityCardPurchaseHandler : Le composant FidelityCardPurchaseHandler a pour rôle de traiter le paiement d'un client qui achèterait un article en utilisant l'argent qu'il a au préalable déposé sur sa carte multi-fidélité.

RefillFidelityHandler : Le composant RefillFidelityHandler a pour rôle de permettre à l'utilisateur de charger de l'argent sur sa carte muli-fidélité pour qu'il puisse l'utiliser en magasin pour faire des achats.

StoreManager : Le composant StoreManager gère les magasins. Il reçoit des demandes du contrôleur StoreController pour enregistrer de nouveaux magasins (par l'interface StoreRegistration), ajouter ou supprimer des employés, changer les heures d'ouverture et le statut des magasins (par l'interface StoreModifier), trouver un magasin (par l'interface StoreFinder) et obtenir les heures d'ouverture et les offres des magasins (par l'interface StoreExplorer). Il utilise l'interface EmployeeFinder pour vérifier si la personne connectée a les droits nécessaires pour modifier les informations ou le staff.

StatsManager: Le composant StatsManager à pour rôle de fournir différentes statistiques. Il expose au CatalogController à travers l'interface StatsExplorer, des statistiques sur le total des avantages/réductions offerts et le total des paiements (par les interfaces AdvantageExplorer DiscountExplorer et PaymentExplorer). Ces statistiques sont disponibles pour un magasin en particulier mais un commerçant partenaire peut aussi voir les statistiques des autres commerçants. De plus, seul un administrateur a accès aux ratios qui concernent les avantages. Cette gestion de droit d'accès est vérifiée lors d'une demande au composant EmployeeManager (par l'interface EmployeeFinder). Il utilise les interfaces EmployeeRegistration et ScheduleModifier pour modifier la liste d'employés et les horaires. Il utilise ScheduleExplorer pour donner différentes informations sur le magasin.

DiscountManager : Le composant à pour rôle de gérer les réduction disponibles dans le système. Il reçoit des demandes de CatalogController, StatsManager et PaymentHandler afin de recevoir les réductions disponibles (par l'interface DiscountExplorer). De plus, il fournit à Store
Controller, à travers l'interface DiscountModifier, la possibilité de modifier les réductions du système. Afin de vérifier que l'utilisateur possède bien les droits pour réaliser les actions de modification ou de récupération des données, il demande au composant EmployeeManager à travers l'interface EmployeeFinder.

AdvantageManager : Le composant à pour rôle de gérer les avantages disponibles dans le système. Il reçoit des demandes de CatalogController, de StatsManager et de VFPAdvantageManager afin de recevoir les avantages disponibles (par l'interface AdvantageExplorer). De plus, le composant fournit à travers l'interface AdvantageModifier le moyen de modifier les avantages proposés par la collectivité territoriale. Seuls les administrateur systèmes possèdent le droit de modifier selon les demandes de la collectivité territoriale les avantages proposés. Afin de vérifier que l'utilisateur possède les droits pour réaliser les actions de modification ou de récupération des données, le composant AdvantageManager interroge le composant EmployeeManager à travers l'interface EmployeeFinder.

EmployeeManager : Ce composant permet la gestion d'employés. Il permet notamment de vérifier les identifiants d'un employé et de retrouver un employé avec son id ou son nom(à l'aide de l'interface EmployeeFinder). Ce component reçoit des requêtes notamment par CustomerAdvertiser,StoreManager et AdvantageManager. Ce composant propose aussi

l'interface EmployeeRegistration qui est utilisé dans StoreManager par l'admin du store pour enregistrer un nouvel employé.

ScheduleManager : Ce composant permet la gestion de l'emploi du temps d'un store. Le StoreManager fait appel à ce composant pour modifier les horaires d'ouvertures (par l'interface ScheduleModifier) et pour voir les informations relatives aux horaires des différents jours (Schedule explorer).

CustomerFidelityManager : Ce composant gère les différentes interactions avec le compte de fidélité d'un client. Il reçoit ainsi des FidelityAccount sur lesquels certaines sommes d'argent ou de points sont à incrémenter et décrémenter. CustomerFidelityManager vérifie ainsi l'intégrité d'une demande (balance non-négative par exemple) avant d'effectuer cette dernière. Ce composant est chargé de traiter les recherches de FidelityAccount par passage d'id. Enfin il vérifie aussi si le client est éligible au programme VFP

PaymentManager : Ce composant est chargé d'enregistrer et de retrouver les différents achats qui ont pu être effectués par un client passé en paramètre ou effectué dans un Store passé en paramètre.

CustomerManager: Le composant CustomerManager gère les données du client (Customer). Il reçoit des demandes de CustomerAccountController pour réaliser l'inscription d'un nouveau client (par l'interface CustomerRegistration). De plus, il répond aux demandes de CustomerAccountController, afin de procéder à la modifications des informations du client à travers l'interface CustomerModifier. Le composant fournit le service d'identification d'un client (par l'interface CustomerFinder) au composants PaymentHandler, VFPTransactionProcessHandler, CustomerAdvertiser. Il expose les informations d'un c Client au CustomerAccountController. Enfin le composant CustomerManager gère les données du client en communiquant avec le CustomerRepository

VI. Scénarios MVP

Consommer un avantage:

- (Application → VFPController → VFPTransactionProcessHandler → CustomerManager → CustomerRepository)
    - (retour positif identification du client CustomerRepository→CustomerManager → VFPTransactionProcessHandler)
- (Vérification de l'identité VFPTransactionProcessHandler → CustomerFidelityManager → FidelityRepository)
    - (retour positif le client est bien VFP FidelityRepository → CustomerFidelityManager → VFPTransactionProcessHandler)
- (Consommation d'un avantage du client VFPTransactionProcessHandler → VFPAdvantageManager → CustomerAdvantageRepository)
    - (Retour positif de la consommation d'avantage jusqu'à Application CustomerAdvantageRepository → VFPAdvantageManager → VFPTransactionProcessHandler → VFPController → Application)

Faire un achat:

- (Application → ClientPaymentController → PaymentHandler → CustomerManager →CustomerRepository)
    - (Retour positif identification du client CustomerRepository→CustomerManager → PaymentHandler)
- (Vérification de l'identité du store PaymentHandler → StoreManager → StoreRepository)
    - (Retour positif identification du store StoreRepository → StoreManager → PaimentHandler)
- (Ajoute des points sur le compte de fidélité PaymentHandler → SettledPurchaseHandler → CustomerFidelityManager → FidelityRepository)
    - (retour positif de la modification des données du repository FidelityRepository → CustomerFidelityManager)
- (Ajoute du paiement dans le repository CustomerFidelityManager —> PaymentManager → PaymentRepository)
    - (Retour positif de l'achat donc du gain de points jusqu' à Application PaymentRepositoryr → PaymentManager → CustomerFidelityManager → SettledPurchaseHandler → PaymentHandler → ClientPaymentController → Application)

Obtenir un cadeau (un achat est nécessaire):

- (Application → ClientPaymentController → PaymentHandler → CustomerManager →CustomerRepository)
    - (Retour positif identification du client CustomerRepository→CustomerManager → PaymentHandler)
- (Vérification de l'identité du store PaymentHandler → StoreManager → StoreRepository)
    - (Retour positif identification du store StoreRepository → StoreManager → PaimentHandler)
- (Utilisation des points pour payer le cadeau PaymentHandler → PointPurchaseHandler → CustomerFidelityManager → FidelityRepository)
    - (retour positif de l'utilisation des points FidelityRepository → CustomerFidelityManager → PointPurchaseHandler → PaymentHandler)
- (Ajoute des points sur le compte de fidélité PaymentHandler → SettledPurchaseHandler → CustomerFidelityManager → FidelityRepository)
    - (retour positif de la modification des données du repository FidelityRepository → CustomerFidelityManager)
- (Ajoute du paiement dans le repository CustomerFidelityManager —> PaymentManager → PaymentRepository)
    - (Retour positif de l'achat jusqu' à Application PaymentRepositoryr → PaymentManager → CustomerFidelityManager → SettledPurchaseHandler → PaymentHandler → ClientPaymentController → Application)

Créer les comptes des commerçants dans le système:

- (Application → StorePartenaireController → StoreManager → EmployeeManager → UserRepository)(retour positif si la personne qui veut créer un compte commerçant est connecté en admin UserRepository → EmployeeManager → StoreManager)(Création du nouveau compte StoreManager → EmployeeManager → StoreManager puis retour jusqu'au front ajout de compte réussi)

Mettre de l'argent sur sa carte fidélité:

- (Application → CustomerAccountController →CustomerManager → CustomerRepository)
    - (Retour positif identification du client CustomerRepository→CustomerManager→CustomerAccountController)
- (On remplie la carte de fidélité CustomerAccountController→RefillFidelityHandler→BankProxy→BankRepository→ BankProxy)
    - (Retour positif si transaction réussi BankProxy→BankRepository→RefillFidelityHandler)
- (On modifie la balance du compte de fidélité RefillFidelityHandler→CustomerFidelityManager→FidelityRepository puis retour jusqu'au front transaction réussi)

Paiement avec sa carte fidélité:

- (Application → ClientPaymentController → PaymentHandler → CustomerManager →CustomerRepository)
    - (Retour positif identification du client CustomerRepository→CustomerManager → PaymentHandler)
- (Vérification de l'identité du store PaymentHandler→ StoreManager → StoreRepository)
    - (Retour positif identification du store StoreRepository → StoreManager → PaymentHandler)
- (Débité la carte de fidélité PaymentHandler→FidelityCartPurchase→CustomerFidelityManager→FidelityRepository)
    - (Retour positif si le client a bien été débité FidelityRepository→CustomerFidelityManager→FidelityCartPurchase →PaymentHandler)
- (Ajoute des points sur le compte de fidélité PaymentHandler → SettledPurchaseHandler → CustomerFidelityManager → FidelityRepository)
    - (retour positif de la modification des données du repository FidelityRepository → CustomerFidelityManager)
- (Ajoute du paiement dans le repository CustomerFidelityManager —> PaymentManager → PaymentRepository)

Gagner de points :

- (Application → ClientPaymentController → PaymentHandler → SettledPurchaseHandler → CustomerFidelityManager → FidelityRepository)
- (Application → ClientPaymentController → PaymentHandler → FidelityCardPurchaseHandler → CustomerFidelityManager → FidelityRepository)

Obtenir des indicateurs chiffrés :

- (vérification des droits d'accés aux indicateurs : Application → CatalogController → StatsManager → EmployeeManager → UserRepository)
    - (si accès autorisé, obtention des indicateurs : UserRepository → EmployeeManager → StatsManager)
      (retour affichage des indicateurs : StatsManager → CatalogController → Application)
    - (si accès non autorisé, message d'erreur : UserRepository → EmployeeManager → StatsManager → CatalogController → Application)

Obtenir indicateurs chiffré sur les cadeaux (discount) fournis:

- (Application → CatalogueController → StatsManager → StoreFinder → StoreRepository)
    - (retour positif d'identification du store StoreRepository → StoreFinder → StatsManager)
- (StatsManager → PaymentManager → PaymentRepository)
    - (retour positif données récupérées PaymentRepository → PaymentManager → StatsManager)
- (retour vers Application des statistiques générée sur les cadeau StatsManager → CatalogueController → Application)

Obtenir indicateurs chiffré sur les avantages fournis:

- (Application → CatalogueController → StatsManager → EmployeeManager → EmployeeRepository)
    - (retour positif identification administrateur EmployeeRepository → EmployeeManager → StatsManager)
- (récupération des données de payement StatsManager → PaymentManager → PaymentRepository)
    - (retour positif données récupérées PaymentRepository → PaymentManager → StatsManager)
- (récupération des données des avantages StatsManager StatsManager → AdvantageManager → AdvantageRepository)
    - (retour positif des données des avantages AdvantageRepository → AdvantageManager → StatsManager)
- (retour vers Application des statistiques générée sur les cadeau StatsManager → CatalogueController → Application)

Alimenter le catalogue de cadeau :

- (vérification des droits d'accès aux modifications : Application → CatalogController → DiscountManager → EmployeeManager → UserRepository)
    - (Retour positif si accès autorisé : UserRepository → EmployeeManager → DiscountManager)
- (Ajout du cadeau DiscountManager → DiscountRepository puis retour positif jusqu'au front-end)

Alimenter le catalogue d'avantages :

- (Application → CatalogController → AdvantageManager → EmployeeManager → UserRepository)
    - (Retour positif si accès autorisé : UserRepository → EmployeeManager→ AdvantageManager)
- ((Ajout de l'avantage AdvantageManager → AdvantageRepository)

VII. Docker Chart