Rapport d'architecture ISA/DevOps

Équipe G

Bezes Bastien

Devictor Pauline

Dubois Quentin

El Kateb Sami

Latapie Florian

24 janvier 2023

Université Côte d'Azur - Polytech Nice – SI4-ISA/DevOps

2022 – 2023

Table des matières

1. Introduction 3

2. Cas d'utilisation 4

Acteurs primaires 4

*Acteurs secondaires ..... **Erreur ! Signet non défini.***

Diagramme de cas d'utilisation 6

3. Objets métiers 9

4. Interfaces 11

5. Composants 15

6. Scénarios MVP 16

7. Docker Chart de la CookieFactory fournie (DevOps) 17

1. Introduction

De nos jours, les consommateurs tendent à effectuer leurs achats en grande surface plutôt que chez les petits commerçants de leur quartier. Afin d'encourager les clients à acheter chez les commerçants de proximité, nous allons développer un système de carte multi-fidélité. Ce système n'est pas exclusif à une ville donnée, mais il est au contraire réutilisable dans différentes villes et différents quartiers. Nous souhaitons aussi devenir le leader sur le marché des cartes de multi-fidélité grâce à notre système.

Le fonctionnement du système est le suivant, les commerçants peuvent rejoindre le programme partenaire et sont identifiables grâce à une vignette présente sur leur porte. Les clients peuvent ensuite acheter chez les commerçants et gagner des points en présentant leur carte multi-fidélité qui seront ensuite échangeables contre divers avantages auprès des commerçants partenaires et de la mairie. En utilisant régulièrement le service, les consommateurs peuvent gagner le statut VFP (Very Faithfull Person) et accéder à des avantages supplémentaires.

Ainsi, le but de ce projet est d'élaborer ce système de carte multi-fidélité, les caractéristiques du système sont résumées dans les points suivants :

Généralités

- La carte ne fonctionne que dans les commerces associés.
- Possibilité de recharger sa carte en ligne avec une faible somme d'argent via carte bleue (maximum : arbitraire).
- Les achats génèrent des points qui pourront être échangés contre des avantages :
    - Lots offerts par la Mairie (Place de parking gratuite pendant un temps donné, Ticket de bus gratuits ou à prix réduit, …)
    - Lots chez les commerçants partenaires

Clients

Un client faisant des achats de manière hebdomadaires en utilisant le service devient VFP et bénéficient d'avantages supplémentaires nommés avantages VFP :

Exemples :

- Lots offerts par la mairie pour les clients VFP (ex : gratuité de 30 minutes de stationnement ou encore un ticket de bus gratuit par jour)
- Catalogue de lots réservés aux VFP chez les commerçants

Lorsqu'un des magasins favoris d'un client change ses horaires, le client est notifié.

Commerçants

Un commerçant peut :

- Obtenir des indicateurs : sur la tendance de consommation de leur client, sur sa contribution au système par rapport aux autres commerçants
- Gérer son catalogue d'offres (classiques et VFP)

Administrateurs système

L'administrateur système peut :

- Obtenir des informations sur les habitudes de consommation des utilisateurs
- Notifier les utilisateurs
    - Lors de la perte de leur statut de VFP
    - Envoyer des offres publicitaires
    - Lancer des sondages de satisfaction auprès des utilisateurs
- Créer des comptes pour les commerçants

2. Cas d'utilisation

Acteurs primaires

Rôle Utilisateur

Jacques – 81 ans

- Débloquer le statut VFP grâce à ses achats hebdomadaires
    - Perdre son statut s'il n'a effectué aucun achat au cours d'une semaine
- Profiter des avantages VFP.

Alison – 29 ans

- Consulter les horaires des magasins
- Gérer sa liste de magasins préférés
- Recevoir les notifications des changements d'horaires de ses magasins préférés
- Recharger sa carte de fidélité en ligne avec une faible somme d'argent

Pierre – 32 ans

- S'abonner au service multi-fidélités
- Profiter des avantages classiques
- Gérer son profil :
    - Ajouter sa plaque d'immatriculation
    - Modifier ses données

Rôle Administrateur système

Franck – 25 ans

- Accéder aux statistiques :
    - Coût des avantages offert par la mairie
    - Volume généré par les usagers en achat “brut”
    - Volume et lieux des cadeaux récupérés
- Accéder à la base de données des utilisateurs pour :
    - Relancer les consommateurs (ex : perte de leur statut de VFP)
    - Envoyer des offres promotionnelles
    - Lancer des sondages de satisfaction aux usagers
- Gérer le catalogue-cadeau d’offres classiques et VFP pour la collectivité (ex : transport ou parking)
- Créer les comptes pour les commerçants

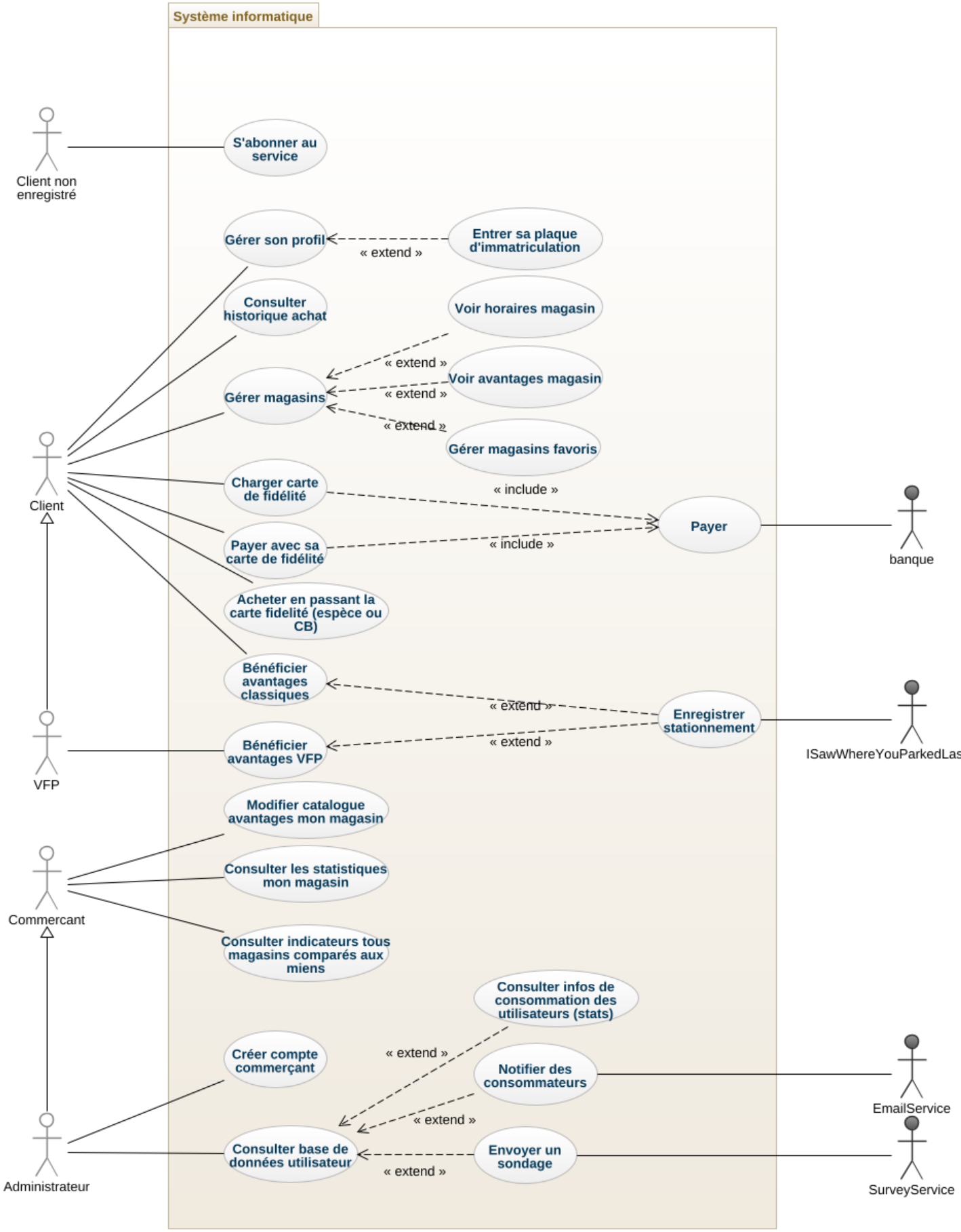
Rôle commerçant partenaire

Laura – 22 ans

- Gérer le catalogue des avantages de son magasin (ajouter, modifier, supprimer) :
    - Quels cadeaux et nombre de points à dépenser pour les obtenir
- Accéder aux statistiques de son magasin :
    - Consulte volume de vente d’usagers utilisant la carte multi-fidélité
- Accéder aux indicateurs sur l’utilisation du programme par rapport aux autres commerçants partenaires

Diagramme de cas d'utilisation

Diagramme au format SVG en ligne



Notre diagramme de cas d'utilisation comporte 5 acteurs principaux :

Les Clients non enregistrés

Lorsque le client arrive sur le site la première fois, il a le rôle de client non enregistré. Il a la possibilité de s'abonner au service.

Les Clients

Ce sont les clients enregistrés, abonnés au service, possédant un compte client et une carte multi-fidélité.

Ils peuvent payer avec leur carte de fidélité, en euros lors de paiements avec leur carte multi-fidélité chez les commerçants, ou en points pour bénéficier de cadeaux.

Lorsque le paiement s'effectue par carte bancaire ou espèces, les clients peuvent utiliser la carte de fidélité pour s'identifier auprès de notre système et ajouter à leur compte les points gagnés grâce à l'achat.

Les utilisateurs ont la possibilité de :

- Consulter les magasins utilisant notre système de fidélité
- Gérer des magasins en favori
- Consulter les informations à propos d'un magasin
    - Les cadeaux proposés
    - Les horaires
- Consulter leur historique d'achat
- Modifier leurs données personnelles

Les VFP

Ce sont des utilisateurs connectés possédant le statut VFP. Ils peuvent bénéficier d'avantages supplémentaires.

Les Commerçants

Ce sont des propriétaires ou employés d'une entreprise ayant rejoint le programme de fidélité, de ce fait, ils peuvent :

- Gérer leur catalogue de cadeau
- Consulter les statistiques de leur magasin pour analyser les habitudes de consommation des utilisateurs du programme de fidélité
- Accéder aux indicateurs sur l'utilisation du programme par rapport aux autres commerçants partenaires

Les administrateurs

Nous avons fait le choix qu'un administrateur soit une spécialisation d'un commerçant, car un administrateur peut faire exactement les mêmes actions qu'un commerçant et il peut faire des actions supplémentaires. Un administrateur système peut donc être vu comme un commerçant du magasin "Mairie" avec plus d'actions. Ses actions supplémentaires permettent de gérer le système et d'avoir accès à plus de données.

Ils peuvent :

- Accéder aux indicateurs statistiques des commerces et des catalogues
- Gérer les demandes d'adhésion au programme de fidélités pour les commerçants (création de compte commerçant)
- Consulter les statistiques de consommation des utilisateurs
- Notifier les consommateurs
- Gérer les sondages, pour cette action, les utilisateurs sont des acteurs secondaires, car ils reçoivent les notifications du serveur

Acteurs secondaires

Banque

Le service externe banque est utilisé dans 2 cas d'utilisation :

- Un utilisateur recharge son compte
- Un utilisateur paye avec sa carte multi-fidélité

IsawWhereYouParkedLastSummer (Service externe parcmètre)

Cette plateforme externe s'occupe du stationnement de la ville. Elle enregistre les voitures stationnées grâce à leur plaque d'immatriculation et la durée du stationnement (date de début et de fin). Elle envoie un rappel à l'utilisateur cinq minutes avant la fin du stationnement et un autre rappel lorsque la période de stationnement est terminée.

- Données envoyées au service via son API : plaque d'immatriculation, date de début, date de fin, un email (ou un autre moyen de notifier l'utilisateur)
- Rappel : Le service externe s'occupe de notifier l'utilisateur

EmailService

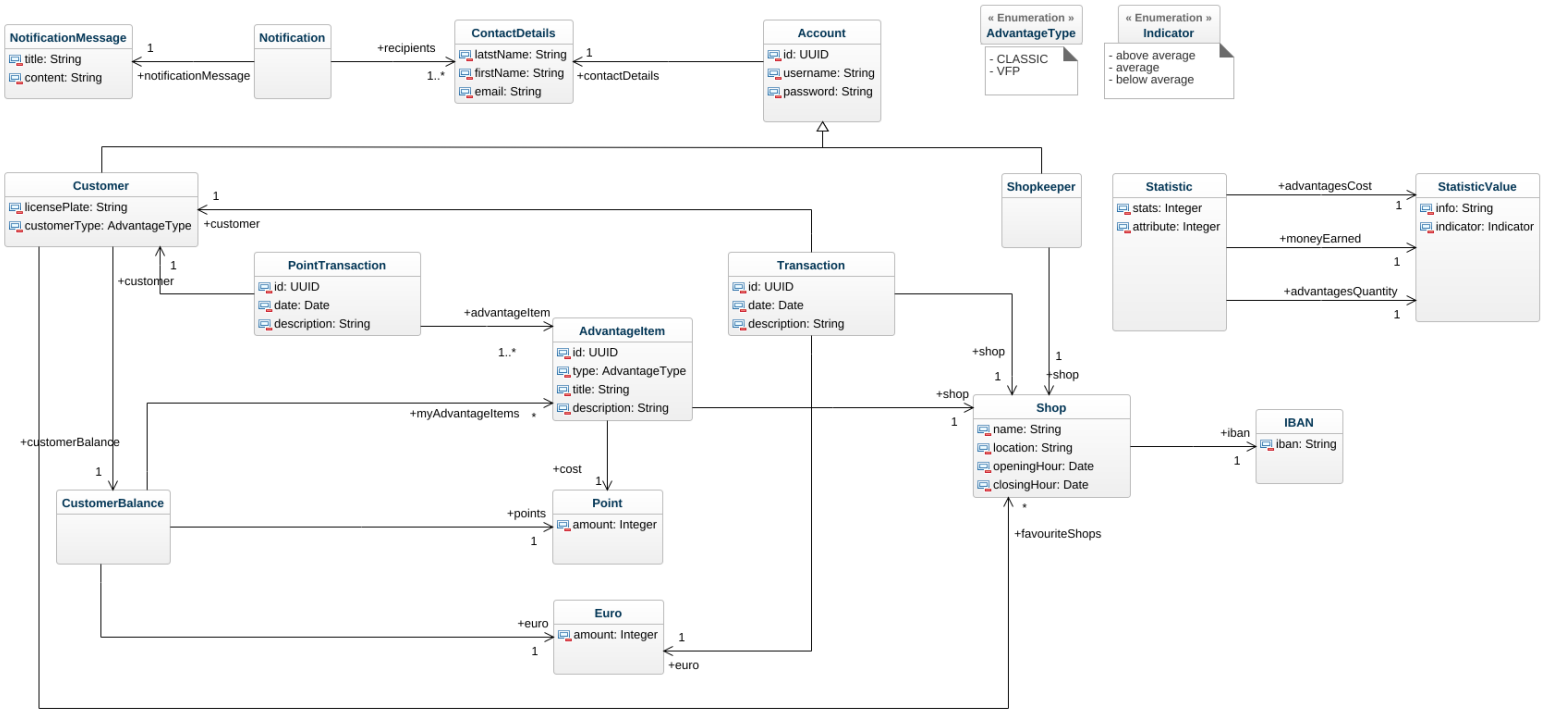
Nous avons décidé d'utiliser un service externe de mail afin de notifier les utilisateurs de notre application. Ce service, comme son nom l'indique, permet d'envoyer un email à l'utilisateur. Nous avons fait ce choix, car cela permet d'uniformiser la notification d'un utilisateur indépendamment de l'interface utilisé pour interagir avec le système (mobile, desktop ou web).

SondageService

C'est un service externe permettant de gérer les sondages et les statistiques associées. Nous avons fait ce choix parce que l'organisation des sondages ne fait pas partie du métier de notre système, nous avons donc préféré externaliser cette fonctionnalité. Ce service est utilisé lors de la création de sondages par un administrateur système.

3. Objets métiers

[Diagramme au format SVG en ligne]



NotificationMessage

Cette classe représente le message d'une notification avec un titre et un contenu.

Notification

Cette classe contient les données nécessaires à l'envoi d'une notification, notamment NotificationMessage (le contenu de la notification) et un ou plusieurs ContactDetails (les destinataires de la notification).

ContactDetails

Classe représentant les informations personnelles d'un client avec son nom, son prénom, son adresse mail.

Account

Classe abstraite représentant les informations basiques d'un compte (identifiant, nom d'utilisateur, mot de passe). Cette classe contient un attribut ContactDetails.

Customer

Classe représentant le compte d'un client dans le système, elle hérite de Account. Il contient toutes les informations additionnelles sur le client nécessaires pour le système telles que sa plaque d'immatriculation, la liste de magasins favoris et un statut (VFP ou classique). Elle contient également un "Customer Balance".

CustomerBalance

CustomerBalance est une classe modélisant la quantité de points, d'euros présents sur le compte d'un client, ainsi que les avantages que le client a achetés en points.

Shopkeeper

Classe représentant le compte d'un commerçant ou d'un administrateur. Nous avons fait ce choix, pour donner suite à la décision que nous avons prise pour le diagramme de cas d'utilisation, qu'un administrateur est un commerçant étendu. Un Shopkeeper est associé à un magasin. Nous avons un compte commerçant par magasin.

Shop

Classe représentant un magasin dans le système. Il contient toutes les informations nécessaires au système comme son nom, son adresse, ses horaires d'ouvertures.

Le magasin possède aussi un IBAN permettant de reverser l'argent au magasin lors d'un paiement par un utilisateur avec sa carte de fidélité.

IBAN

Cette classe modélise un IBAN par un seul attribut stocké dans un String. Cela permet de vérifier que le format est valide quand on le stocke.

Transaction

Cette classe représente une transaction effectuée entre un client et un commerçant. Elle contient une quantité d'euros exprimée par la classe Euro et d'autres informations complémentaires tels qu'un identifiant, une date et une description.

PointTransaction

Cette classe correspond à une transaction en point. Cet objet est utilisé lors de l'achat d'un avantage en point par l'utilisateur et lors de l'utilisation d'un avantage. Cette classe contient bien évidemment un client (Customer) et une liste d'avantage (AvantageItem).

AdvantageItem

AdvantageItem correspond à la représentation d'un avantage client avec un certain type AdvantageType, Classic ou VFP. Il possède aussi des informations telles qu'un identifiant, un titre et une description.

Point

Point est une classe représentant les points gagnés ou dépensés par les clients dans le système.

Euro

Euro est une classe représentant des euros dans le système.

Statistics

Cette classe représente un objet contenant l'ensemble des données envoyées à un commerçant ou un administrateur système. Ces données sont stockées dans des StatisticValue. La classe Statistics continent trois attributs de type StatisticValue :

- moneyEarned: l'argent total dépensé par les client utilisant la carte multi-fidélité
- advantagesQuantity: la quantité en volume d'avantages délivrée par le magasin
- advantagesCost: le coût total des avantages délivrés par le magasin

StatisticValue

Cette classe représente une donnée avec un entier codant la valeur et un indicateur représentant le positionnement du magasin par rapport à la moyenne des autres magasins sous la forme d'un type énuméré (Indicator).

4. Interfaces

CustomerFinder

```
public interface CustomerFinder {
    Customer login(String username, String password) throws
CustomerNotFoundException;
    Customer find(UUID id) throws CustomerNotFoundException;
}
```

Cette interface permet de récupérer un Customer.

CustomerModifier

```
public interface CustomerModifier {
    Customer signup(String username, String password, ContactDetails
contactDetails) throws CustomerAlreadyExistsException;
    Customer update(Customer customer) throws CustomerNotFoundException;
    // Customer updateBalance(Customer customer, CustomerBalance balance)
throws CustomerNotFoundException;
}
```

Cette interface permet de modifier un Customer.

CustomerNotifier

```
public interface CustomerNotifier {
    Notification notify(Predicate<Customer> filter, NotificationMessage
notificationMessage);
    Notification notify(List<Customer> customers, NotificationMessage
notificationMessage);
}
```

Cette interface permet de notifier un ou plusieurs clients par email.

ShopFinder

```
public interface ShopFinder {
    List<Shop> find(String name);
    Shop find(UUID id);
}
```

Cette interface permet de récupérer un Shop.

ShopModifier

```
public interface ShopModifier {
    Shop create(String name, String location, DateTime openingHour, DateTime
closingHour, String bankAcount);
    Shop update(Shop shop);
}
```

Cette interface permet de modifier un Shop.

TransactionFinder

```
public interface TransactionFinder {
    Transaction findEuro(UUID id);
    List<Transaction> findEuro(Shop shop);
    List<Transaction> findEuro(Customer customer);

    PointTransaction findPoint(UUID id);
    List<PointTransaction> findPoint(Shop shop);
    List<PointTransaction> findPoint(Customer customer);
}
```

Cette interface permet de récupérer une transaction.

StatisticsFinder

```
public interface StatisticsFinder {
    Statistic find(Shop shop);
    List<Statistic> find();
}
```

Cette interface permet de récupérer les statistiques associées à un magasin ou les statistiques globales.

Payment

```
public interface Payment {
    EuroTransaction createTransaction(Euro amount, Customer customer);
    EuroTransaction createTransaction(Euro amount, List<AvantageItem>
avantages, Customer customer);
```

```
    PointTransaction createTransaction(Point amount, List<AvantageItem>
avantages, Customer customer);
}
```

Cette interface permet d'effectuer une transaction en point ou en euro.

ShopKeeperFinder

```
public interface ShopKeeperFinder {
    ShopKeeper login(String username, String password) throws
ShopKeeperNotFoundException;
    ShopKeeper find(UUID id) throws ShopKeeperNotFoundException;
}
```

Cette interface permet de récupérer un ShopKeeper

ShopKeeperModifier

```
public interface ShopKeeperModifier {
    ShopKeeper createAccount(String username, String password, ContactDetails
contactDetails) throws ShopkeeperAlreadyExistsException;
    ShopKeeper update(ShopKeeper shopkeeper) throws
ShopkeeperNotFoundException;
}
```

Cette interface permet de modifier un ShopKeeper

AdvantageFinder

```
public interface AdvantageFinder {
    List<AdvantageItem> find(Shop shop);
    AdvantageItem find(UUID id);
}
```

Cette interface permet de récupérer des avantages soit en fonction d'un magasin ou directement via l'identifiant unique d'un avantage.

AdvantageModifier

```
public interface AdvantageModifier {
    AdvantageItem create(String title, Point amount, Shop shop, String
description);
    AdvantageItem update(AdvantageItem advantageItem);
}
```

Cette interface permet d'ajouter ou du modifier un avantage.

Bank

```
public interface Bank {
    boolean pay(Customer customer, Euro price) throws PaymentException;
}
```

Cette interface permet de réaliser un paiement en euro.

Survey

```
public interface Survey {
    void newSurvey(List<Customer> customers);
}
```

Cette interface permet de gérer les sondages.

Park

```
public interface Park {
    void parkForFree(String licensePlate, DateTime startTime, DateTime
endTime);
    void parkForDuration (String licensePlate, DateTime startTime, int
minutesDuration)
}
```

Cette interface permet de communiquer avec le service Externe de parking permettant d'offrir des minutes de parking gratuites.

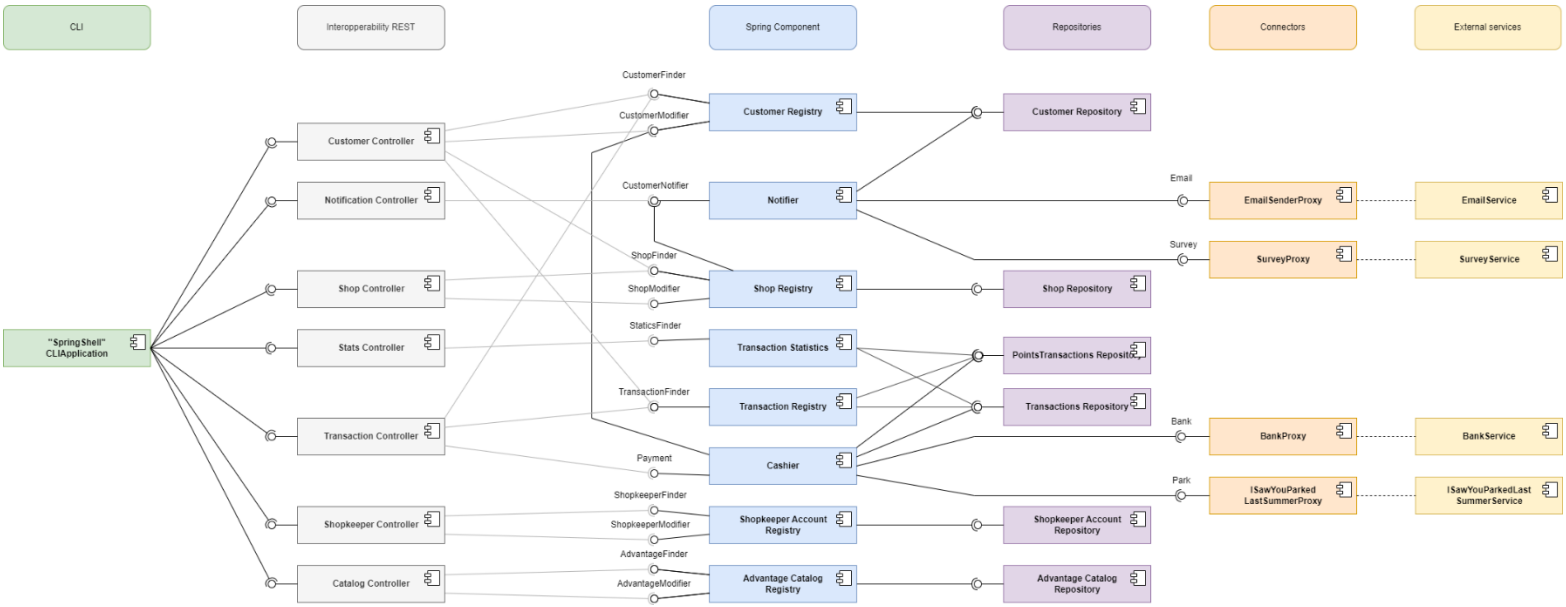
Email

```
public interface Email {
    void send(List<Customer> customer);
    void send(Customer customer);
}
```

Cette interface permet d'envoyer un email à un ou plusieurs utilisateurs.

5. Composants

[Diagramme au format SVG en ligne]



Explication des composants

Customer registry

Le composant Customer registry gère les Customer. Il reçoit des demandes du Customer Controller pour ajouter/modifier les clients, récupérer des clients. Il reçoit également des demandes de récupération de clients via le Transaction Controller.

ShopKeeper registry

Le composant ShopKeeper registry gère les ShopKeeper. Il reçoit des demandes du ShopKeeper Controller pour ajouter/modifier les gérants de magasins et pour récupérer des gérants de magasins.

Advantage registry

Le composant Advantage registry gère les AdvantageItem. Il reçoit des demandes du Advantage Controller pour ajouter/modifier les avantages des magasins et pour récupérer les avantages des magasins.

Transaction registry

Le composant Transaction registry gère les Transaction et PointTransaction. Il reçoit des demandes du TransactionController pour ajouter de nouvelles transactions. Il reçoit pareillement des demandes du CustomerController pour récupérer toutes les transactions par rapport à un client.

Transaction statistics

Le composant Transaction statistics gère les statistiques globales ou les statistiques associée à un magasin. Il reçoit des demandes du Stats Controller.

Cashier

Le composant Cashier s'occupe de gérer les paiements en euros et en points. Il reçoit des demandes du Customer Controller et du ShopKeeper Controller. Il communique toujours avec le Customer registry afin d'ajouter les points générés par les achats. Dans certains cas, d'autres informations du client sont modifiés ; notamment lors de l'utilisation d'un avantage, le AdvantageItem est supprimé du client et lors d'un paiement avec la carte-multi-fidélité la quantité en euros est retirée du compte client.

Il communique aussi avec le proxy de la banque pour les rechargements de la carte multi-fidélité en euros et pour un paiement avec la carte mutli-fidélité chez un commerçant.

Il communique également avec le proxy du service ISawYouParkedLastSummer lors de l'achat d'un avantage impliquant le stationnement pour voiture.

Notifier

Le composant Notifier permet de notifier un ou plusieurs clients, actuellement uniquement par email. Il reçoit des demandes du Notification Controller.

Shop Registry

Le composant Shop Registry reçoit des demandes du Shop Controller pour ajouter/modifier les magasins et récupérer des magasins.

6. Scénarios MVP

Inscription du client

CLI → Customer Controller → Customer Registry → Customer Repository

Achat d'un avantage par un client

CLI → Transaction Controller → Cashier → Transaction Registry → PointTransactions Repository

→ Customer Registry → Customer Repository

Payement en liquide ou carte bleue

Présentation de la carte multi-fidélité auprès d'un commerçant et utilisation d'un avantage (le paiement se fait de manière externe, en liquide ou en carte bleue)

CLI → Transaction Controller → Customer Modifier → Customer Repository

→ Cashier → TransactionRegistry → Transaction Repository

→ TransactionRegistry → PointsTransaction Repository

→ Customer Modifier → Customer Repository

7. Docker Chart de la CookieFactory fournie (DevOps)

La commande **docker compose up** lance les trois sous-conteneurs dans l'ordre suivant :

- bank-system
- tcf-server
- tcf-cli

Une fois tout cela lancé, on peut se connecter sur la CLI à l'aide de la commande **docker attach cli**. (Attaché directement au conteneur sans utiliser d'adresse IP ou de port)

Nous avons obtenu les adresses IP du sous-réseau à l'aide de la commande **docker network inspect simpletcfs_default** et les autres informations à partir du fichier docker-compose et des docker files.

Nous avons mis le nom des services dans le diagramme, car, dans le sous-réseau, les différents conteneurs sont accédés par le nom de service.