Polytech Nice-Sophia
Département Sciences Informatiques
930 Route des Colles
06410 Biot
04 92 96 50 50
www.polytechnice.fr

ISA

INTRODUCTION TO SOFTWARE ARCHITECTURE

Carte d'infidélité

Enseignants encadrants

Anne-Marie PINNA-DERY
Sébastien MOSSER

-

César COLLÉ
Basil DALIÉ
Loïck MAHIEUX

I. Table des matières

II. Introduction

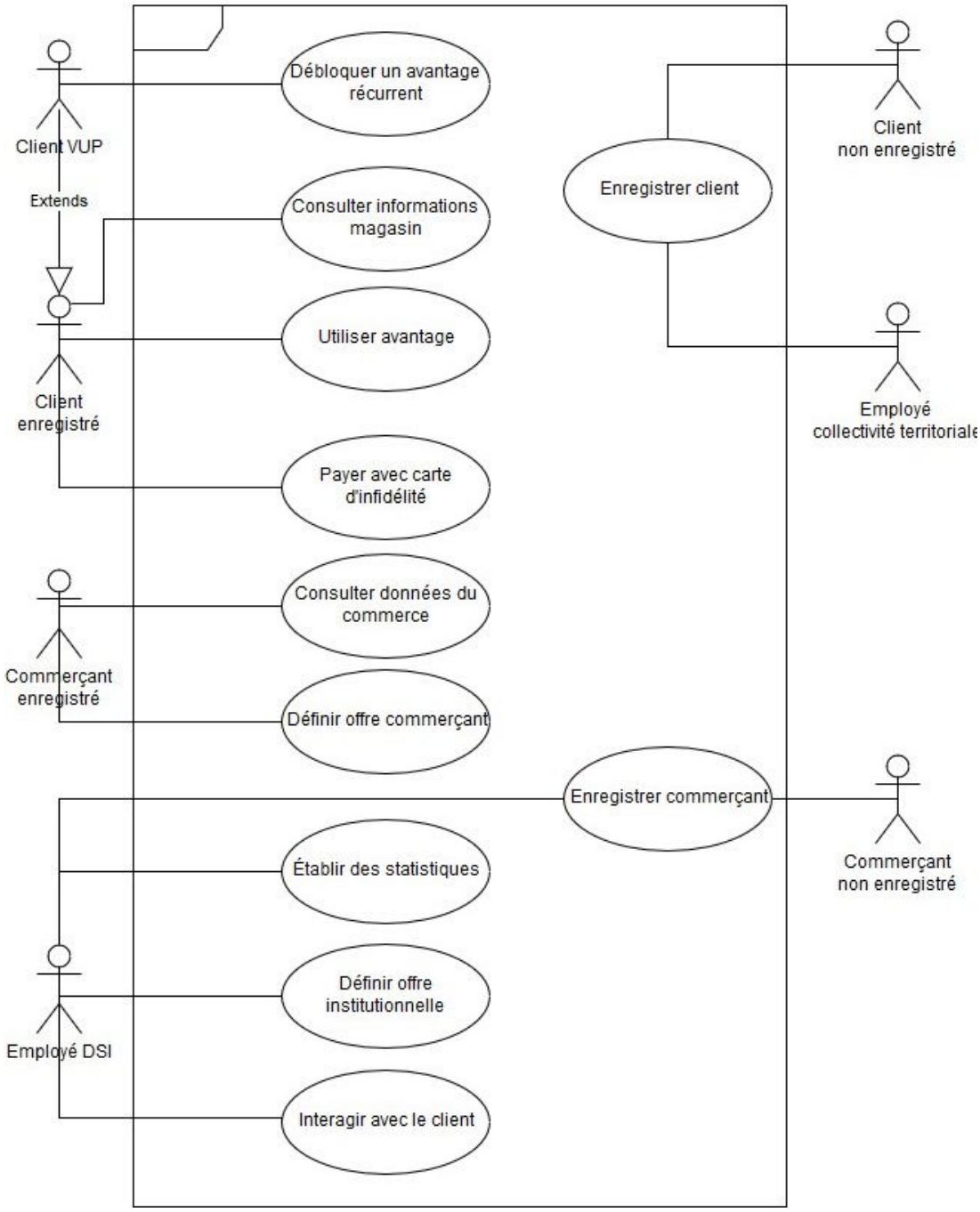
Les logiciels d'aujourd'hui deviennent de plus en plus complexe au niveau des fonctionnalités qu'ils offrent. Les logiciels informatiques sont devenus des bastions grouillant de fonctionnalités les plus diverses et variées. Il en vient alors que pour bâtir ces édifices, il est nécessaire alors d'établir un plan d'attaque. C'est alors qu'intervient le rôle de l'architecte logiciel.

Nous allons à travers ce document, traiter les différentes étapes d'évolution de notre architecture d'une application permettant de créer des cartes d'infidélités. Nous verrons dans un premier temps les scénarios que nous avons dégagé des spécifications clients, puis nous verrons par la suite la vue fonctionnelle, la vue de développement et enfin la vue de déploiement de notre système.

III. Vue fonctionnelle

A. Diagramme de cas d'utilisation

1. Diagramme de cas d'utilisation de haut niveau

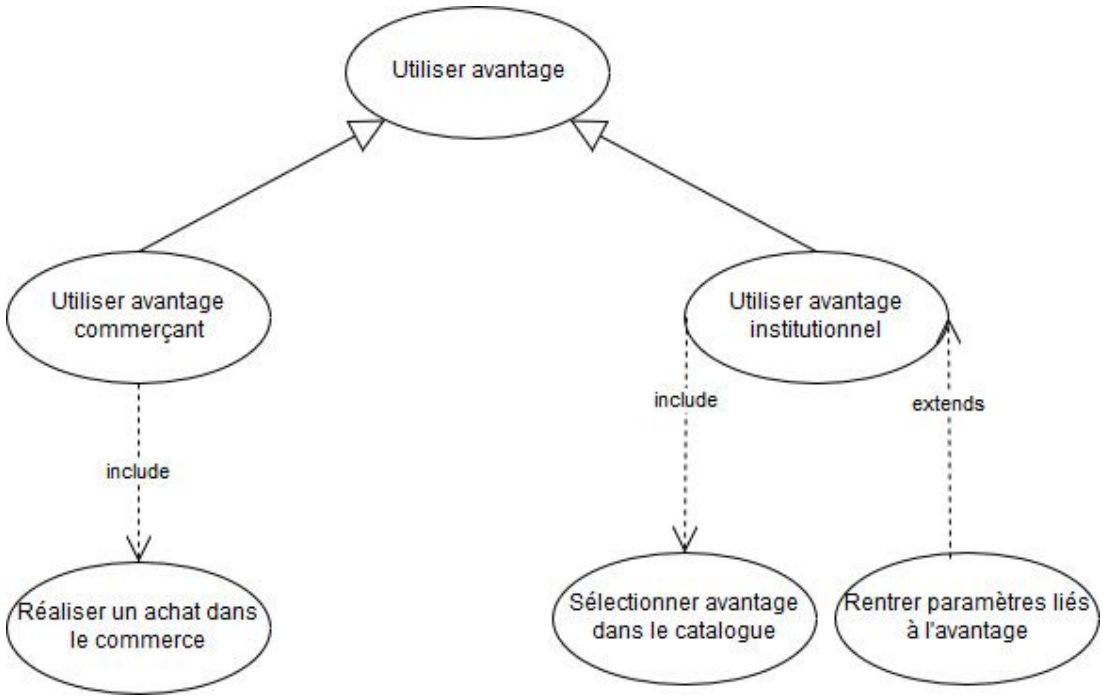


2. Description des acteurs

Au cours de la phase d'analyse, nous sommes parvenus à isoler huit acteurs participant au fonctionnement du système :

- **Client enregistré** : il s'agit d'un client abonné au service de carte d'infidélité, ce dernier peut consulter des informations sur un magasin (par exemple les horaires), et utiliser un avantage en dépensant des points gagnés à travers ses achats. Il peut également payé avec sa carte d'infidélité dans un magasin, s'il a préalablement chargé de l'argent sur sa carte.
- **Client VUP** : il s'agit d'un client enregistré qui a obtenu le statut de VUP. En plus des fonctionnalités précédemment citées, il a accès à un catalogue plus vaste d'avantage, et peut débloquer un avantage récurrent qui reste valide dans qu'il conserve son statut de VUP.
- **Commerçant** : il s'agit d'un commerçant enregistré sur le service, il peut définir des offres sur son commerce, et a accès à des données concernant l'impact du service son son commerce et sa position par rapport aux autres partenaires.
- **Employé DSI** : il s'agit d'un employé de la direction des systèmes d'informations de la collectivité territoriale (mairie, collectivité de commune, etc.) qui utilise le service de carte d'infidélité. Il a la responsabilité d'enregistrer les commerçant qui veulent rejoindre le service, ajouter des offres institutionnelles, établir des statistiques sur l'utilisation du service, et communiquer avec les clients (relances, sondages, offres promotionnelles)
- **Employé collectivité territoriale** : il s'agit d'un employé de la collectivité territoriale qui emploie le service, il peut enregistrer les clients qui souhaite s'inscrire.
- **Client non enregistré**, et **commerçant non enregistré** : il s'agit des clients et commerçants de la zone couverte par le service qui ne sont pas enregistrés.

3. Cas d'utilisation : Utiliser avantage



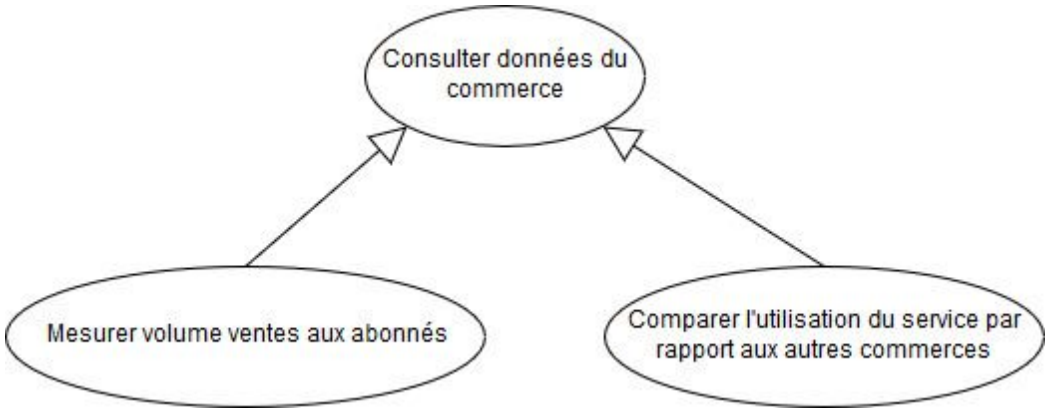
Ce cas d'utilisation concerne un **Client enregistré,** il se décline en 3 scénarios d'utilisation:

le client peut vouloir utiliser un avantage propre à un commerce, il doit néanmoins avoir préalablement réalisé un achat dans le commerce concerné. Pour utiliser son avantage, le client doit confier sa carte au commerçant afin que ce dernier la scanne. Une fois l'opération effectuée, le nombre de points disponibles sur la carte est diminué en fonction du prix de l'avantage.

Il peut également utiliser un avantage institutionnel proposé par la collectivité territoriale employant le service (par exemple des places de parking gratuites), pour cela il doit sélectionner dans son catalogue l'avantage souhaité, une fois l'opération effectuée l'avantage reste actif pour une durée pré-déterminée lors de sa définition. Certains avantages peuvent nécessiter la saisie de paramètres, par exemple, un client possédant une place de parking offerte doit pouvoir saisir sa plaque d'immatriculation au moment de l'activation. S'il s'agit d'un avantage récurrent débloqué par un client possédant le statut de VUP, le nombre de point reste inchangé. Sinon, le nombre de points disponibles sur la carte est diminué en fonction du prix de l'avantage.

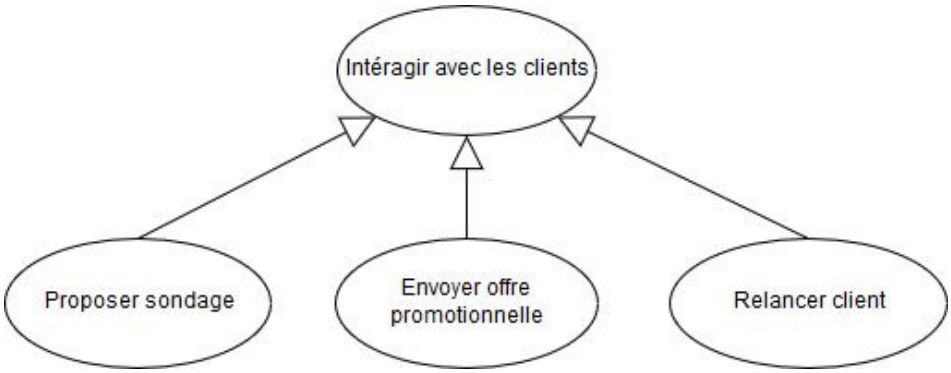
Enfin, sur l'interface du catalogue il a accès à un formulaire dans lequel il peut soumettre des idées d'avantages non proposés dans le catalogue.

4. Cas d'utilisation : Consulter données du commerce



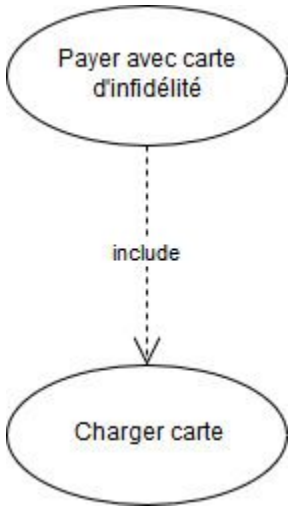
Ce cas d'utilisation concerne un commerçant enregistré. Celui-ci dispose d'une interface lui permettant d'accéder à un ensemble de données chiffrées concernant l'impact du service sur son commerce, par exemple le montant des ventes associé aux clients enregistrés. Celle-ci propose également une vue permettant de comparer sa participation au service avec les autres commerces partenaires.

5. Cas d'utilisation : Interagir avec les clients



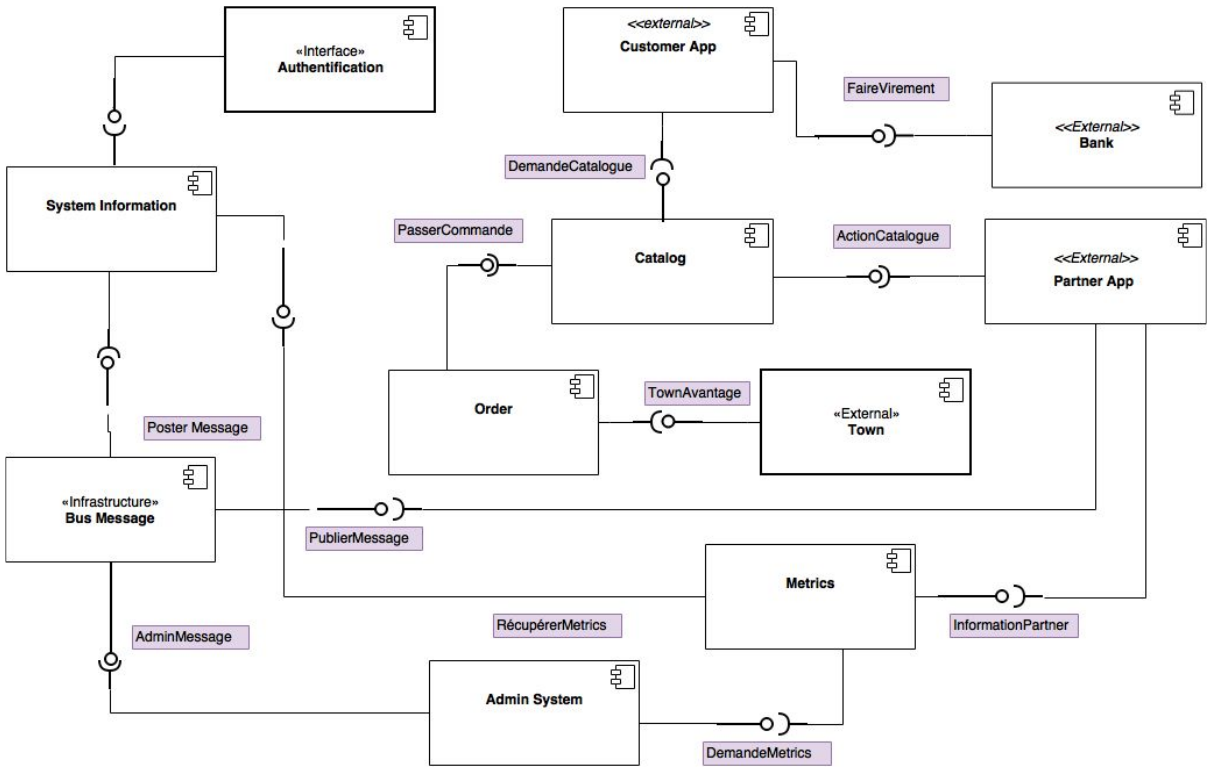
Ce cas d'utilisation concerne la communication entre la direction des systèmes d'information et les clients enregistrés. Celui-ci peut envoyer des sondages aux clients (par exemple pour évaluer leurs habitudes d'utilisation ou leur satisfaction), envoyer des offres promotionnelles. Enfin, à partir de l'exploitation des données des utilisateurs, il peut choisir de relancer certains clients.

6. Cas d'utilisation : Payer avec carte d'infidélité



Concernant le paiement en caisse au moyen de la carte d'infidélité, le client doit préalablement placer une somme d'argent sur sa carte à travers une interface web. Ensuite il confie sa carte au commerçant pour que ce dernier la scanne avec son smartphone.

B. Diagramme de composants



Interface DemandeMetrics:

```
int nombreDemandeAvantage(Partner partner)
int frequentionClient(int idCustomer)
int listeAchatClient(int idCustomer)
```

Interface AdminMessage:

```
void faireSondage(Array<Client> clients)
void EnvoyerEnqueteSatisfaction(Array Client)
```

Interface DemandeInformation:

```
void demanderHoraire(Partner s)
```

Interface TownAvantage:

```
// exemple de town avantage
void ajouterVoitureParking(Customer customer,string matricule)
void getBusTicket(Customer customer)
```

Interface ActionCatalogue:

```
void ajouterAvantage()
void ajouterAvantageVUP()
```

Interface PosterMessage:

```
void broadcastMessage(Array<Customer> array, string msg)
```

Interface DemandeCatalog:

// This interface allow to user to consult the whole list of partner shop.

```
Array<Article> demanderCatalog(Shop s)
Array<Article> demanderCatalogAll()
```

Interface PasserCommande:

```
void acheterItem(Gift gift, int amount)
```

Interface FaireVirement:

```
void rechargerCarte(int amount)
```

Interface InformationPartner :

```
void changerInformationPartner(Shop s)
int volumeVentes(Customer c)
Array<Gift> getGifts(Shop s)
```

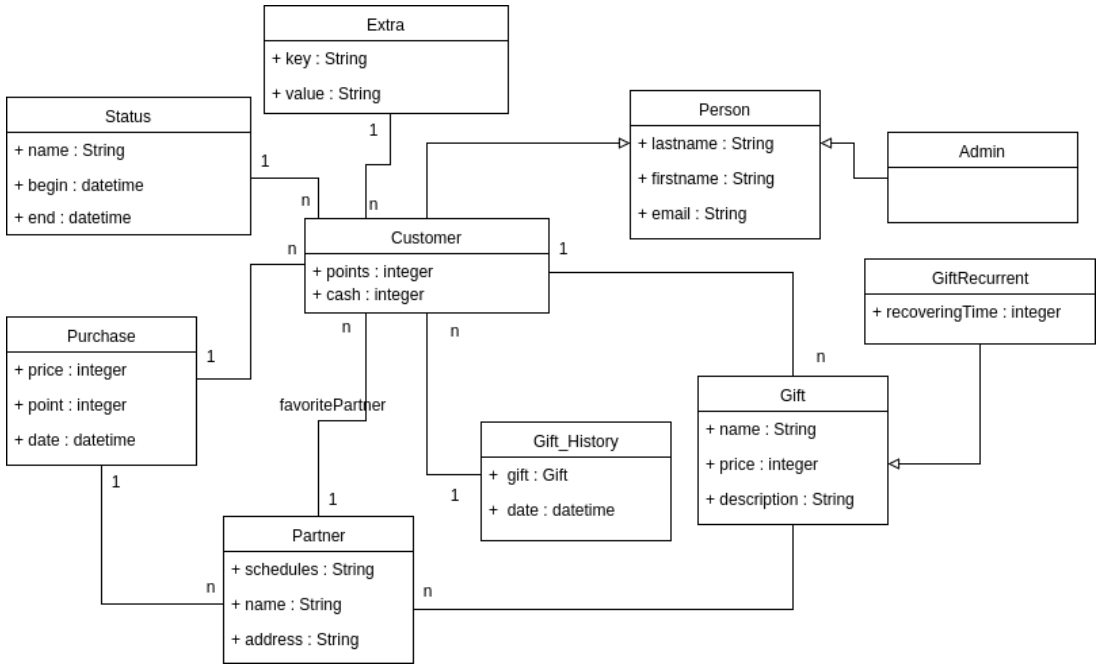
Nous considérons que l'architecture de notre système doit avant tout mettre en avant le côté backend du logiciel. C'est pour cela que toute l'interface entre notre système et les utilisateurs est définie comme "external". Cela nous permet donc de passer outre cette partie tout en montrant aux stakeholder la relation entre notre système et ces dernières.

Nous avons privilégié la création d'une infrastructure nous permettant de pouvoir communiquer avec nos utilisateurs via un certains nombre de device et en utilisant des protocoles de communications différents.

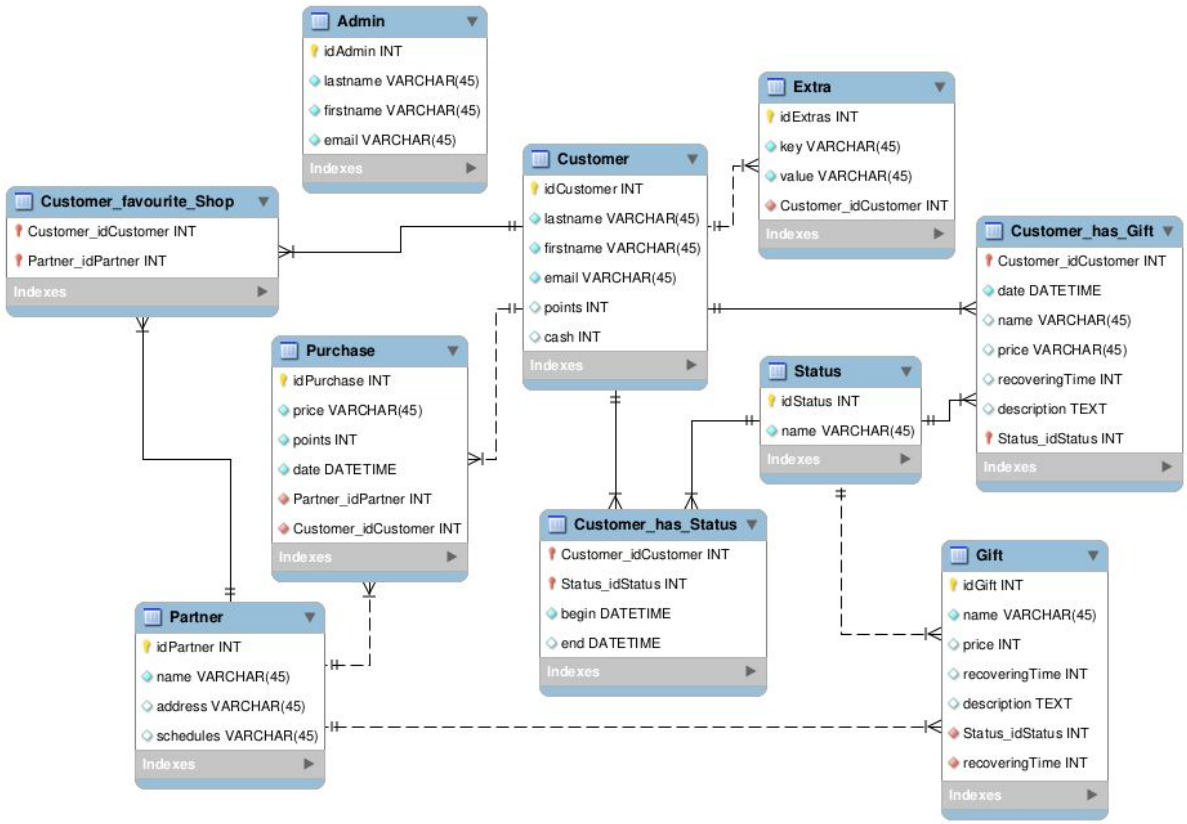
Nous avons aussi fait en sorte de regrouper le plus possible les interfaces entre elles, en évitant l'antipatron Swiss Army Knife, c'est à dire de donner trop de responsabilité à nos interfaces. Il nous a fallu alors créer des composant permettant aussi de communiquer facilement avec les composants externes afin de ne pas avoir des composants avec trop de fonctionnalités.

IV. Vue développement

A. Diagramme de classe des objets métiers du système



B. Modèle relationnel de stockage



C. Explicitation du mapping objet-relationnel

Pour construire notre modèle relationnel des données, nous sommes partis du diagramme de classe et objets métiers. Pour représenter les données de façon persistante dans une base de données relationnelle, nous avons dû faire des choix de représentation des relations entre tous ces objets.

Dans notre système, nous avons une relation de hiérarchie entre les «Person», les «Admin» et les «Customer», nous avons choisi de les représenter dans 2 tables différentes. En effet, nous avons choisi cette solution car elle nous permet de bien séparer ces deux objets qui au final seront utilisés dans un but différent. Grâce à cette séparation, nous pouvons gérer indépendamment les données. En effet, les comptes des administrateurs auront certainement moins de variations dans le futur que les clients car notre système est orienté sur l'infidélité des clients. Il sera également plus simple d'accéder aux données et de les gérer car il n'y aura pas de jointures complexes à effectuer pour retrouver la totalité des informations.

Pour représenter les partenaires («Partner») favoris des clients, nous avons choisi de faire une table d'association. Il est ainsi facile de récupérer les partenaires favoris pour un client mais également les clients qui ont comme favori un partenaire afin de, par exemple, les notifier lors d'un changement d'horaires ou faire de la publicité ciblé.

Pour ce qui est de l'héritage d'un cadeau récurrent («GiftRecurrent»), nous avons fait le choix de mettre toutes les informations des deux types de cadeaux dans la même table, étant donné que ce sont tous des cadeaux identiques. En effet, nous pouvons considérer les cadeaux récurrents comme des cadeaux particuliers avec un prix de 0€ et considérer les cadeaux normaux comme des cadeaux récurrents particuliers avec un temps de récupération de 0.

Pour certains cadeaux, il sera nécessaire de stocker d'autres informations sur les clients. Nous avons donc fait le choix de rajouter une table « Extra » qui permet de stocker des informations en clé/valeur pour un client donné.

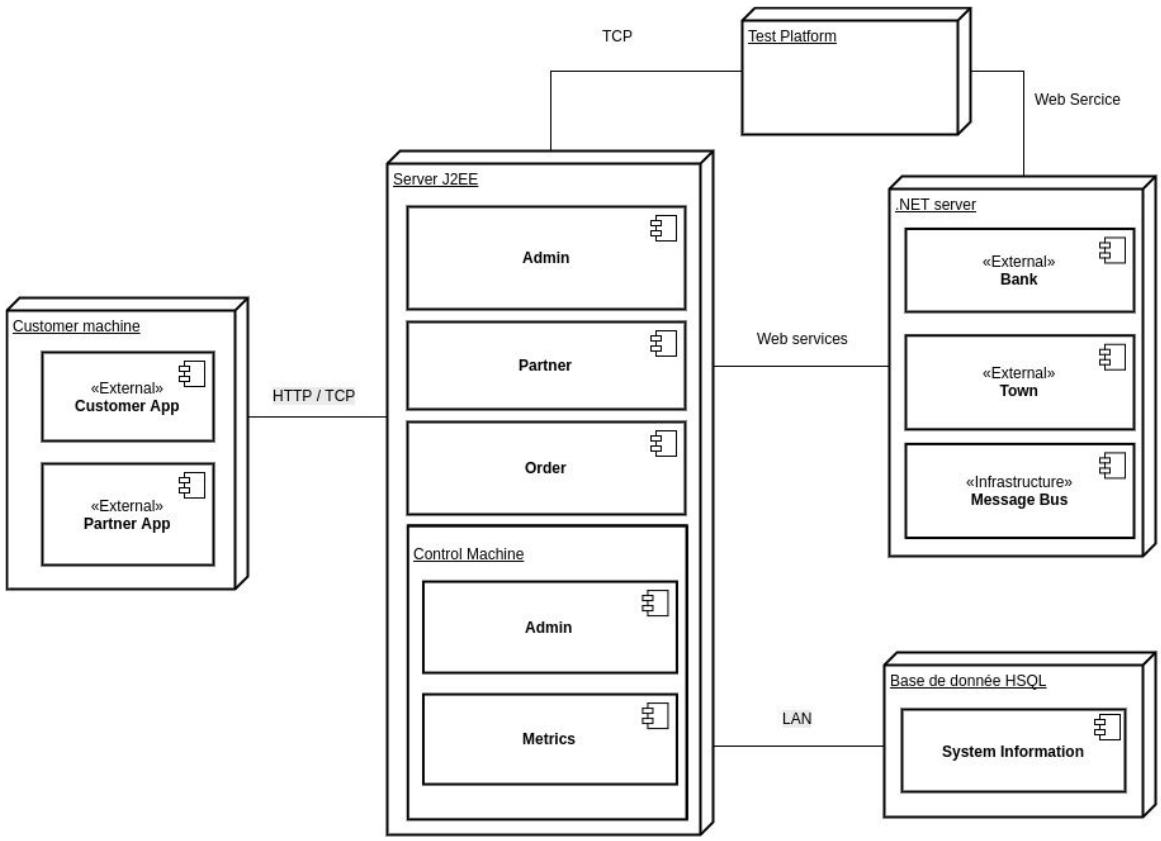
Cependant, il est nécessaire d'avoir des catalogues différents selon le type de client (VUP ou non). Pour représenter ceci, nous avons ajouté une table «Status». Un client pourra donc avoir plusieurs «Status» grâce à une table d'association. Un cadeau sera attaché à un « Status » grâce à une clé étrangère. Ceci va donc nous permettre une plus grande flexibilité dans la gestion des cadeaux. Il faudra donc effectuer une jointure entre les «Status» du client et le catalogue pour récupérer les cadeaux qui lui sont accessibles. Cette table de «Status» nous permet également de stocker l'historique des «Status» obtenus par un client pour éventuellement détecter un changement d'habitude chez le client.

Nous avons choisi de stocker un historique d'achat de cadeau afin de toujours garder une trace des achats de fidélité des clients mais également de garantir la véracité des données en cours d'utilisation des cadeaux en cas de modifications. Cependant, nous avons également une table «Purchase» qui permet de stocker l'historique de tous les achats effectués par le client chez les partenaires. Cette table va donc nous permettre de vérifier quels achats ont été effectués par quel client et éventuellement faire des propositions personnalisées ou vérifier qu'un client a déjà acheté chez ce partenaire (pour le récupération de cadeau entre autre).

V. Vue déploiement

A. Diagramme de déploiement des composants sur les serveurs physiques

On donne le diagramme de déploiement suivant :



On identifie alors plusieurs systèmes de déploiement dans notre cas. Dans un premier temps il apparaît que tous les composants intervenant dans la réalisation des spécifications données par le client sont présents. Nous n'observons aucun besoin matériel véritable.

Nous observons néanmoins que la base de donnée peut être représenter une contrainte dans le sens où elle a une limite d'espace de stockage de l'information. Mais comme le système n'est déployé que pour une ville en particulier on peut fixer facilement une limite.

Pour ce qui est du serveur J2EE. Les entrées sont faites par les commerçant/utilisateurs via l'application, on peut facilement estimer que du point de vue concurrence notre système pourra subir des perturbations. Il est donc important de garder cette variable lors du déploiement et donc ajuster la puissance de la machine qui héberge notre serveur J2EE.

VI. Conclusion

Pour conclure, cette première phase du projet nous aura premièrement permis d'identifier l'ensemble des acteurs et scénarios d'utilisation, et de découper le projet en plusieurs composants indépendants, communiquant via des interface. Nous avons aussi observé que les interface que proposent nos composants permettent de retracer le plus fidèlement les actions décrites dans le diagramme de cas d'utilisation. Nous avons également construit un diagramme de classe, ce qui nous permet d'organiser l'ensemble des données à stocker en éléments indépendants et de décrire leur mise en relation.

Enfin, nous avons fait l'inventaire des moyens matériels et logiciels participant au fonctionnement du service et les protocoles par lesquels ils communiquent entre eux.

Finalement, nous avons constaté que ce rapport est enfaite un document qu'il faut bâtir avec le client et selon certains contraintes. Il est donc important d'apporter de la cohérence et d'avoir une vision cohérente avec celle du client.