POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

Rapport architecture ISA :

Système de carte multi-fidélités

Equipe C

Ben Aissa Nadim
Boubia Marouane
Zoubair Hamza
Saissi Omar
El Garmit Youssef

SOMMAIRE

1. Introduction

Nous voulons développer une architecture logicielle pour un système de carte multi-fidélité pour les clients des commerçants, qui pourrait être utilisée dans différentes zones géographiques. Notre système vise l'incitation à l'achat dans les commerces partenaires, avec des avantages pour les clients fidèles, tels que des offres gratuites dans les commerces partenaires ou des avantages de la collectivité territoriale associée en échange de points. Il permet également de charger de petits montants sur la carte pour des achats chez les partenaires. Les clients les plus fidèles peuvent également obtenir le statut de Very Faithfull Person (VFP) qui permet de débloquer des avantages institutionnels supplémentaires.

Nous supposons que :

- Un internaute peut créer un compte.
- Un client peut se connecter pour consulter le catalogue cadeau, les horaires des magasins et charger sa carte de fidélité.
- Un client peut ajouter un magasin en favori pour être notifié des offres promotionnelles et des changements d'horaires de ce magasin.
- Le statut VFP du client change automatiquement en fonction du nombre d'achats par semaine.
- Le chargement de la carte fidélité utilisable pour n'importe quel achat, dans les commerces partenaires dans la même zone géographique.
- Chaque achat effectué permet d'ajouter des points de fidélité proportionnelle au montant d'achat. Ces points seront utilisés pour bénéficier des offres proposées.
- Un administrateur crée les comptes des commerçants dans le système.
- Un administrateur peut alimenter le catalogue cadeau au niveau de toute l'application en précisant le cadeau et le nombre de points que vaut le cadeau et en définissant les offres classiques et VFPs.
- Un administrateur peut ajouter des sondages.
- Un commerçant peut aussi alimenter le catalogue cadeau de son magasin.
- Les commerçants et les administrateurs peuvent accéder aux statistiques d'achats des autres magasins.

2. Cas d'utilisation :

a) Acteurs primaires :

- Un **Guest** peut :
    - Créer un compte.

- Un **Client** représenté par Alison peut :
    - Consulter le catalogue des cadeaux
    - Recharger sa carte fidélité
    - Ajouter un magasin en favoris
    - Payer ses commandes

- Un **Client VFP** représenté par Jacque et Pierre peut :
    - Jacques : Profiter des avantages VFP, notamment de la gratuité des transports en communs
    - Pierre : Profiter des avantages VFP, notamment de la gratuité des parkings pour une courte durée

- Un **Administrateur** représenté par Franck qui travaille à la DSI de la mairie d'Antibes, qui peut :
    - Lancer des sondages de satisfaction aux usagers.
    - Alimenter le catalogue cadeau.
    - Créer les comptes "commerçants" dans le système.

- Un **Commerçant** représenté par Laura vendeuse à une boulangerie peut :
    - Gérer le catalogue des cadeaux
    - Consulter les statistiques des autres magasins

b) Acteurs secondaires :

- Un **System Notifier** utilisé à fin de :
    - Notifier le persona Alison lorsqu'un magasin change ces horaires par exemple si elle a ajouté ce magasin en favoris.

- Un **Bank System** utilisé pour le persona Alison :
    - Pour payer la commande.
    - Pour recharger sa carte de fidélité.

- **IsawWhereYouParkedLastSummer** utilisé :
    - Lorsque le parking est choisi comme avantage par le persona Pierre.

c) Diagramme de cas d'utilisation :

La figure ci-dessous montre le diagramme de cas d'utilisation élaboré de notre système de carte multi-fidélité:

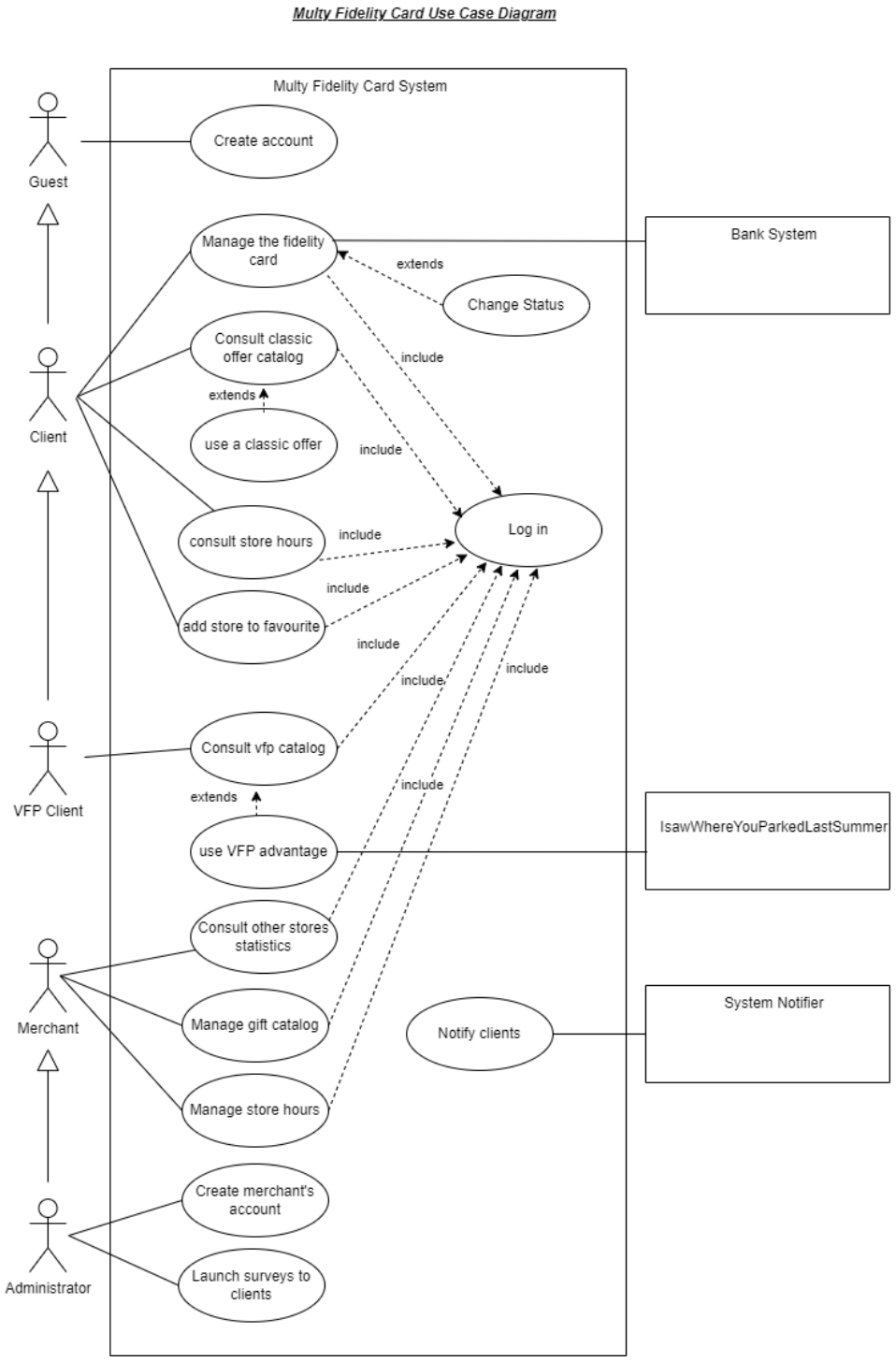

Diagramme de cas d'utilisation du "Multi Fidelity Card System"

Explication du diagramme de cas d'utilisation :

Nous avons décidé de distinguer le client du client VFP étant donné qu'un client normal ne peut pas privilégier des offres VFP :

- Un client normal (supposant qu'il est connecté) peut simplement consulter le catalogue classic, recharger sa carte fidélité avec une "faible" somme d'argent à l'aide de sa carte bleue, ajouter des magasins comme favoris pour être notifié de ces changements d'horaires et des offres promotionnelles.
- Un client VFP peut en plus consulter et utiliser les offres VFP.

Nous avons également le marchand qui peut gérer le catalogue cadeau au niveau de son magasin seulement et peut ainsi accéder aux statistiques des autres magasins. Nous avons aussi l'administrateur qui à plus de responsabilités donc peut gérer le catalogue cadeau à une plus grande échelle comme les cadeaux pour le transports ou parkings, créer les comptes "commerçant" dans le système et interagir avec les utilisateurs en faisant des sondages.

3. Objets métiers :

La figure ci-dessous montre le diagramme de classes d'objets métiers de notre système de carte multi-fidélité:

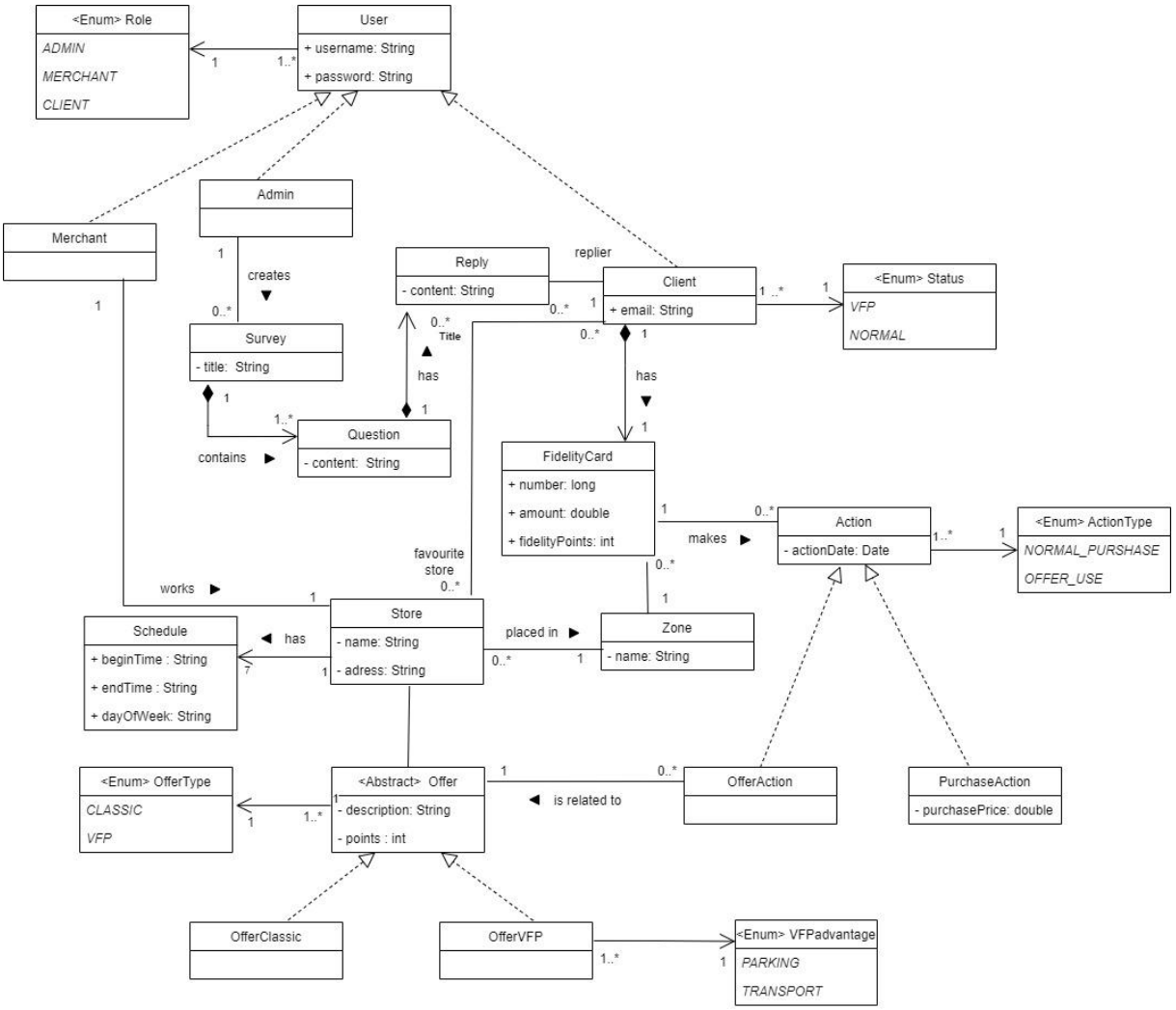

Diagramme de classes du "Multi Fidelity Card System"

Explication du diagramme de classes :

Notre système de carte multi-fidélité est divisé par zone géographique. Nous avons créé la classe **Zone** et chaque carte de fidélité (représentée par la classe **FidelityCard**) est liée à une seule zone géographique.

La classe User est la classe mère des classes **Client**, **Admin** et **Merchant**. Elle est liée à l'enum Rôle qui représente les trois rôles possibles qu'un utilisateur peut avoir. Nous avons ainsi le commerçant, qui est représenté par la classe **Merchant** est lié à un seul magasin (représenté par la classe **Store**). Nous avons également le Client, qui hérite de **User** est

associé à l'enum Status qui représente les deux statuts possibles pour un client : statut NORMAL ou VFP, ce qui nous permet de savoir si le client peut consulter et utiliser les offres **VFPOffer** et **ClassicOffer**.

Comme chaque client veut connaître les horaires de chaque magasin, nous avons ajouté la classe **Store**. Elle est liée à la classe **Schedule** qui contient les horaires pour un jour de la semaine et chaque magasin peut publier plusieurs offres (présentées par la classe **Offer**).

Afin de connaître l'éligibilité d'un client à un statut VFP à travers le nombre d'achats hebdomadaires, nous avons ajouté la classe **Action** qui enregistre la date de chaque action effectuée par la carte de fidélité du client dans un magasin donné. De plus, pour aider les administrateurs à obtenir des informations sur les habitudes de consommation des utilisateurs et les commerçants à avoir des indicateurs sur l'utilisation du programme auprès des autres partenaires, nous avons ajouté deux types d'actions à travers l'enum **ActionType** : **NORMAL_PURCHASE** ou **OFFER_USE**. Nous avons ainsi créé deux classes qui héritent de la classe **Action** : **OfferAction**, liée à la classe **Offer**, et **PurchaseAction**, qui représente le montant d'achat effectué dans un magasin.

Nous avons également ajouté une classe **Survey** pour les sondages créés par les administrateurs. Chaque sondage est composé de plusieurs questions et chacune de ces questions contient les réponses des clients.

4. Interfaces :

IUserAuth: Cette interface permet aux utilisateurs de se connecter ainsi que créer les comptes clients et commerçants.

```
public interface IUserAuth {
  boolean login(String username, String password) throws AccountNotFoundException;
  Client createClientAccount(Client client) throws EmailValidationException, PasswordValidationException,
AccountExistsException ;
  Merchant createMerchantAccount(Merchant merchant) throws AccountExistsException;
}
```

IStatusVFPChange: Cette interface permet de modifier le statut du client selon le nombre de commandes faites par semaine.

```
public interface IStatusVFPChange {
  void changeToVFPClient(UUID clientID);
  void changeToNormalClient(UUID clientID);
}
```

IUserFinder: Cette interface permet de rechercher des utilisateurs selon plusieurs critères.

```
public interface IUserFinder {
  User findUserById(UUID userID);
  List<String> findUserMailsSubscribedToStore(Store store);
}
```

IFidelityCardUse: Cette interface permet de créer et utiliser la carte de fidélité en payant lors d'un achat et en chargeant la carte avec un montant donné.

```
public interface IFidelityCardUse {
  boolean depositAmount(FidelityCard fidelityCard, double amount);
  boolean pay(FidelityCard fidelityCard, double amount) throws InvalidPaymentException;
  boolean createCard(FidelityCard fidelityCard);
}
```

IFidelityPointsModifier: Cette interface permet d'ajouter des points de fidélités lors d'un achat, et de retirer des points lors de la récupération d'un cadeau.

```
public interface IFidelityPointsModifier {
  void addFidelityPoints(FidelityCard card, double purchasingPrice);
  void retrieveFidelityPoints(FidelityCard card, Offer offer);
}
```

IClientHistory: Cette interface permet d'avoir l'historique du client en achat pour vérifier son éligibilité au cadeau ainsi que calculer le nombre d'achat par semaine pour le statut VFP

```
public interface IClientHistory {
  boolean verifyCanHaveGift(Client client,Store store);
  int calculateNbPurchaseLastWeek(Client client);
}
```

IActionSaver: Cette interface permet de sauvegarder les achats et cadeaux des clients .

```
public interface IActionSaver {
  void savePurchaseAction(FidelityCard card);
  void saveGiftAction(FidelityCard card);
}
```

ISalesGiftsStats: Cette interface permet aux commerçants et administrateurs d'obtenir des indicateurs sur l'utilisation du programme auprès des autres partenaires.

```
public interface ISalesGiftsStats {
  List<Action> consultStoreSalesLastMonth(Store store);
  List<Action> consultStoreGiftsLastMonth(Store store);
}
```

IStoreManager: Cette interface permet d'ajouter un nouveau magasin qui correspond au commerçant associé, ainsi que la possibilité de modifier les horaires du magasin.

```
public interface IStoreManager {
  void addStore(Store store);
  void editStoreScheduler(UUID storeID, Scheduler scheduler) throws StoreNotFoundException;
}
```

IStoresExplorator: Cette interface permet d'afficher les magasins disponibles dans une zone géographique donnée.

```
public interface IStoresExplorator {
  List<Store> findStoresByGeoZone(GeoZone zone);
}
```

IStoreFavourite: Cette interface permet d'ajouter des magasins favoris aux clients et de les consulter pour qu'ils soient notifiés lors de l'ajout d'une offre ou la modification des horaires.

```
public interface IStoreFavourite {
  List<Store> findFavouriteStoresForClient(Client client);
  boolean addStoreToFavourite(Store store, Client client);
}
```

IOfferManager: Cette interface permet de gérer les offres avec l'ajout, la suppression et la modification des offres des commerçants.

```
public interface IOfferManager {
  void addOffer(Offer offer);
  void editOffer(UUID offerID, Offer offer) throws OfferNotFoundException;
  void deleteOffer(UUID offerID) throws OfferNotFoundException;
}
```

IOffersExplorator: Cette interface permet de rechercher et de consulter les différentes offres disponibles dans les magasins partenaires.

```
public interface IOffersExplorator {
  List<Offer> findAllOffersByStore(Store store);
  List<ClassicOffer> findClassicOffersByStore(Store store);
  List<VFPOffer> findVFPOffersByStore(Store store);
  Offer findOfferById(UUID id);
}
```

IOfferUse: Cette interface permet d'utiliser les différentes offres disponibles par exemple lors de la récupération d'un cadeau, d'un ticket de transport gratuit ou du parking.

```
public interface IOfferUse {
  void receiveGift(FidelityCard card);
  void getFreeTransport(FidelityCard card);
  void getParking(FidelityCard card, String vehicleRegistration);
}
```

ISurveyCreator: Cette interface permet de créer de nouveaux sondages pour les utilisateurs.

```
public interface ISurveyCreator {
  void createSurvey(Survey survey);
  void respondToServey(Reply reply);
}
```

ISurveyExplorator: Cette interface permet de consulter les sondages créés.

```
public interface ISurveyExplorator {
  List<Survey> viewSurvies();
}
```

IBank: Cette interface permet de communiquer avec le service externe de la banque.

```
public interface IBank {
  boolean pay(FidelityCard card, double amount) throws PaymentException;
  boolean deposit(FidelityCard card, double amount);
}
```

IParking: Cette interface permet de communiquer avec "*IsawWhereYouParkedLastSummer*".

```
public interface IParking {
  void sendDataChrono();
}
```

INotifier: Cette interface permet la notification par email aux utilisateurs lors du changement horaire et ajout d'une nouvelle offre à un magasin favoris.

```
public interface INotifier {
  void notifyAll(List<String> to, String subject, String text);
}
```

5. Composants :

UserService: Le composant gère les utilisateurs du système. Il reçoit des demandes du *CartController* pour l'authentification des utilisateurs à travers l'interface *IUserAuth*, et pour changer le statut à travers *IStatusChange* en utilisant l'interface IClientHistory. Il reçoit également une demande du Notifier pour trouver les utilisateurs qui ont marqué un magasin comme favori à travers *IUserFinder*.

ActionHistoryService: Le composant gère les actions effectuées par les clients. Il reçoit des demandes du *ConsumptionStatisticsController* pour obtenir des indicateurs sur l'utilisation du programme à travers *ISalesGiftsStats*. Il permet également de sauvegarder les achats et les cadeaux récupérés à la demande de *FidelityCardService* et *OfferService* à travers *IActionSaver*. Il permet également de connaître l'historique du client pour vérifier son éligibilité au statut VFP à la demande de UserService, ainsi que son éligibilité à recevoir un cadeau dans un magasin à la demande de *OfferService*, à travers *IClientHistory*.

FidelityCardService: Le composant gère la carte de fidélité d'un client. Il reçoit des demandes du *FidelityCardController* pour payer et charger la carte à travers *IFidelityCardUse*, et utilise *IActionSaver* pour sauvegarder le montant de l'achat associé. Il permet également de modifier les points de fidélité à la demande de *OfferService* à travers *IFidelityPointsModifier*.

OfferService: Le composant permet, à la demande du *OfferController*, de gérer (ajouter, supprimer et modifier) des offres à travers *IOfferManager*, de consulter les offres classiques et VFP et de les utiliser à travers *IOfferUse* et *IOfferExplorer*. Il utilise l'interface *INotifier* pour notifier les utilisateurs lors de l'ajout d'une offre dans un magasin favori.

StoreService: Le composant permet, à la demande du *StoreController*, d'ajouter des magasins et de modifier les horaires à travers *IStoreManager*. Il permet également de consulter les magasins dans le système à travers *IStoreExplorer*. Il utilise l'interface *INotifier* pour notifier les utilisateurs lors du changement des horaires d'un magasin favori.

SurveyService: Le composant permet, à la demande du *StoreController*, d'ajouter et de consulter des sondages à travers *ISurveyAdder* et *ISurveyStatisticsExplorer*.

La figure ci-dessous montre le diagramme de composants de notre système de carte multi-fidélité:

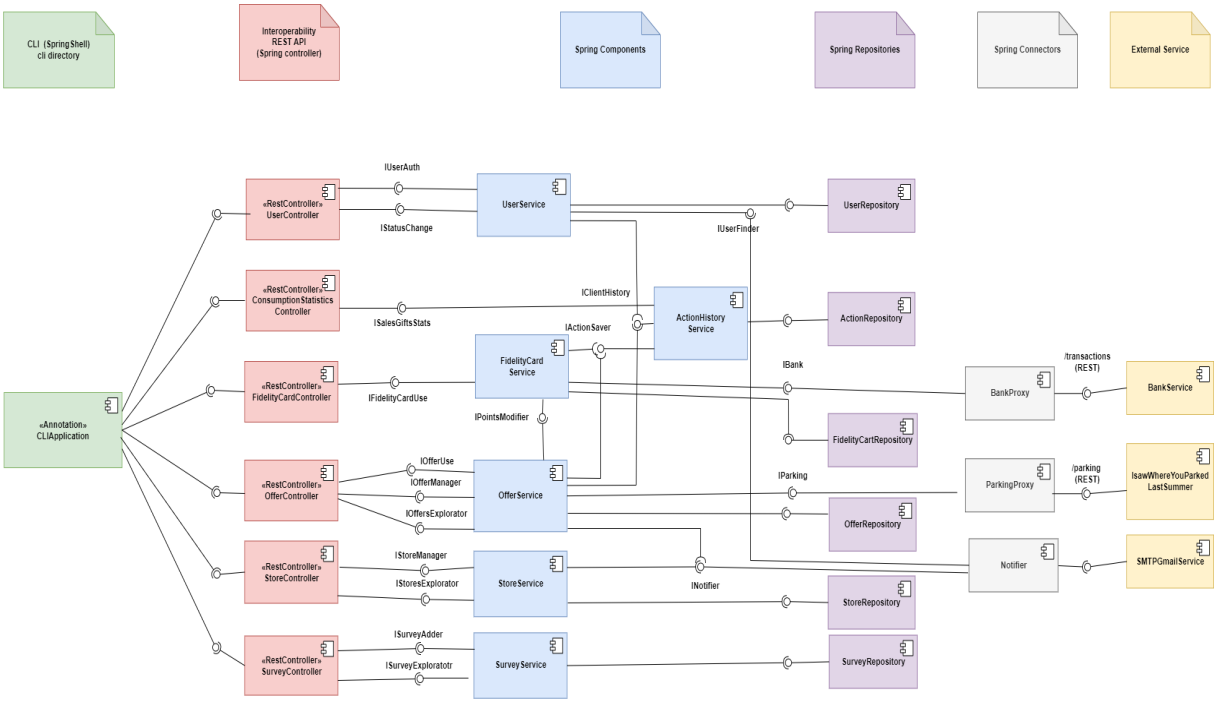

Diagramme de composants du "Multi Fidelity Card System"

6. Scénarios MVP :

- **L'inscription d'un client :**
  (CLI -> UserController -> UserService -> UserRepositroy)

- **Charger la carte de fidélité avec un montant :**
  (CLI -> FidelityCardController -> FidelityCardService -> BankProxy -> BankService)

- **Explorer les offres :**
  (CLI -> OfferController -> OfferService -> OfferRepository)

- **Ajouter / supprimer des offres :**
  (CLI -> OfferController -> OfferService -> OfferRepository)

- **Utiliser une offre classique :**

  ***1**- Enlever les points de fidélité :*
  (CLI -> OfferController -> OfferService -> FidelityCardService -> FidelityCardRepository **)**

  ***2**- Enregistrer l'action dans la BD :*
  (CLI -> OfferController -> OfferService -> ActionsHistoryService -> ActionRepository pour enregistrer l'action dans la BD)

- **Utiliser une offre VFP (Parking) :**

  ***1**- Appeler le service externe liée au Parking :*
  (CLI -> OfferController -> OfferService -> ParkingProxy-> IsawWhereYouParkedLastSumme)

  ***2**- Enlever les points de fidélité :*
  (CLI -> OfferController -> OfferService -> FidelityCardService -> FidelityCardRepository)

  ***3**- Enregistrer l'action dans la BD :*
  (CLI -> OfferController -> OfferService -> ActionsHistoryService -> ActionRepository pour enregistrer l'action dans la BD)

- **Modifier les horaires d'un magasin :**

  ***1**- Sauvegarder les changements d'horaire du magasin :*
  (CLI -> StoreController -> StoreService -> StoreRepository)

2- *Notifier les clients qui ont marqué ce magasin comme favori :*
(CLI -> StoreController -> StoreService -> INotfier-> SMTPGmailService pour notifier le client)

- **<u>Explorer les magasins :</u>**
  (CLI -> StoreController -> StoreService -> Store Repository)

- **<u>Ajouter un sondage :</u>**
  (CLI -> SurveyController -> SurveyService -> SurveyRepository)

- **<u>Changer le statut d'un client en VFP :</u>**
  (CLI -> UserController -> UserService -> UserRepository)

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