Rapport d’architecture

Nathael Nogues

Salah Dahmoul

Yassine Tijani

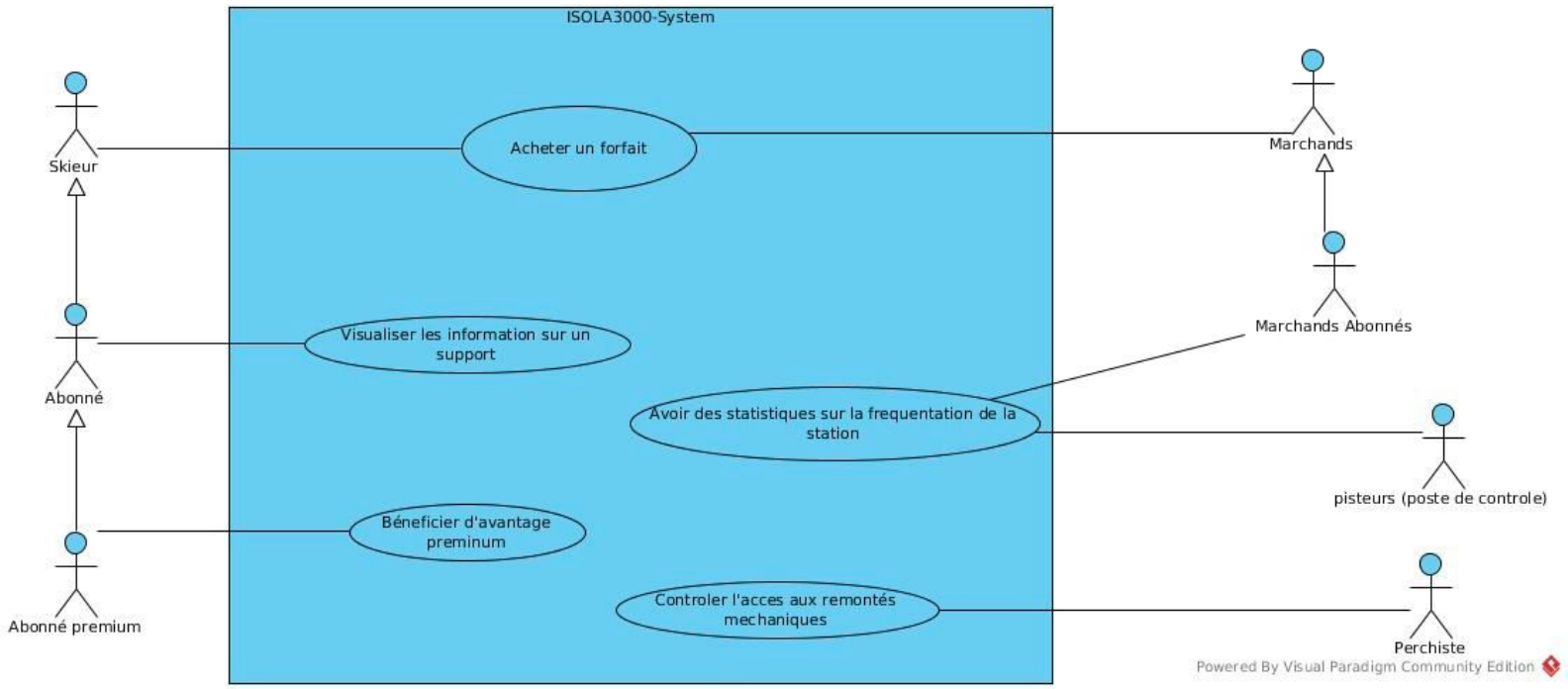
Université Nice Sophia Antipolis

POLYTECH NICE-SOPHIA

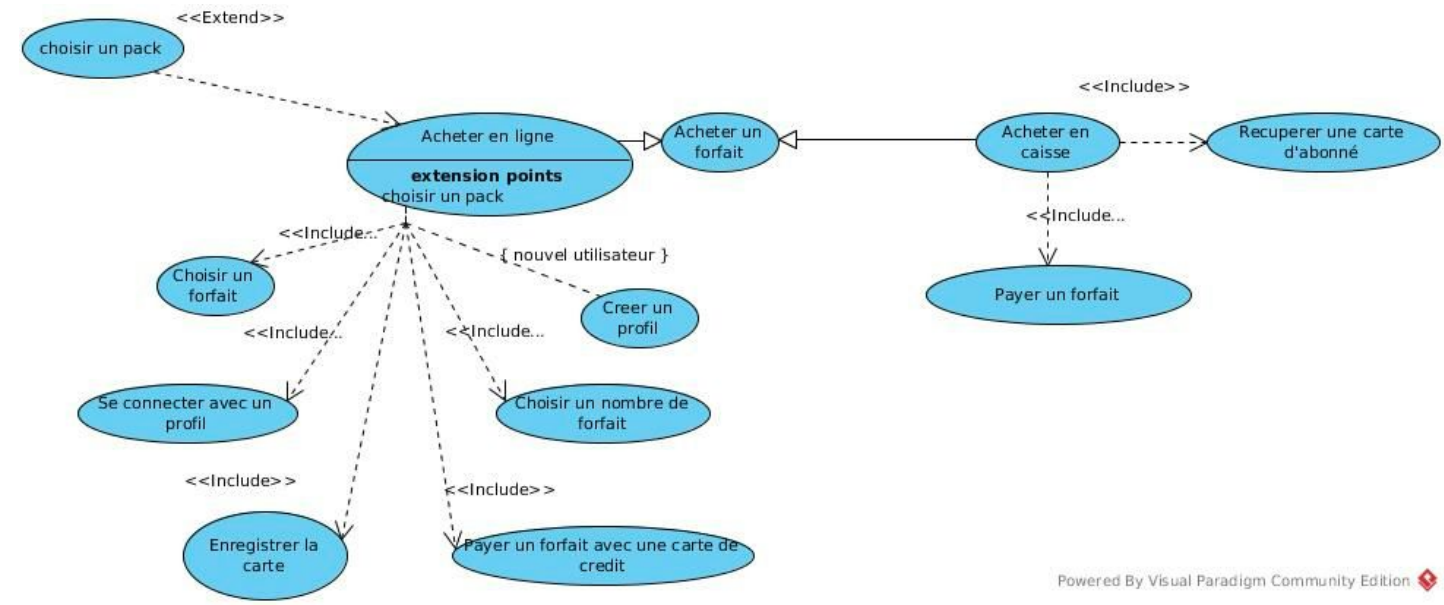
Sommaire

I - Vue fonctionnelle

1.2 - Cas d'utilisations

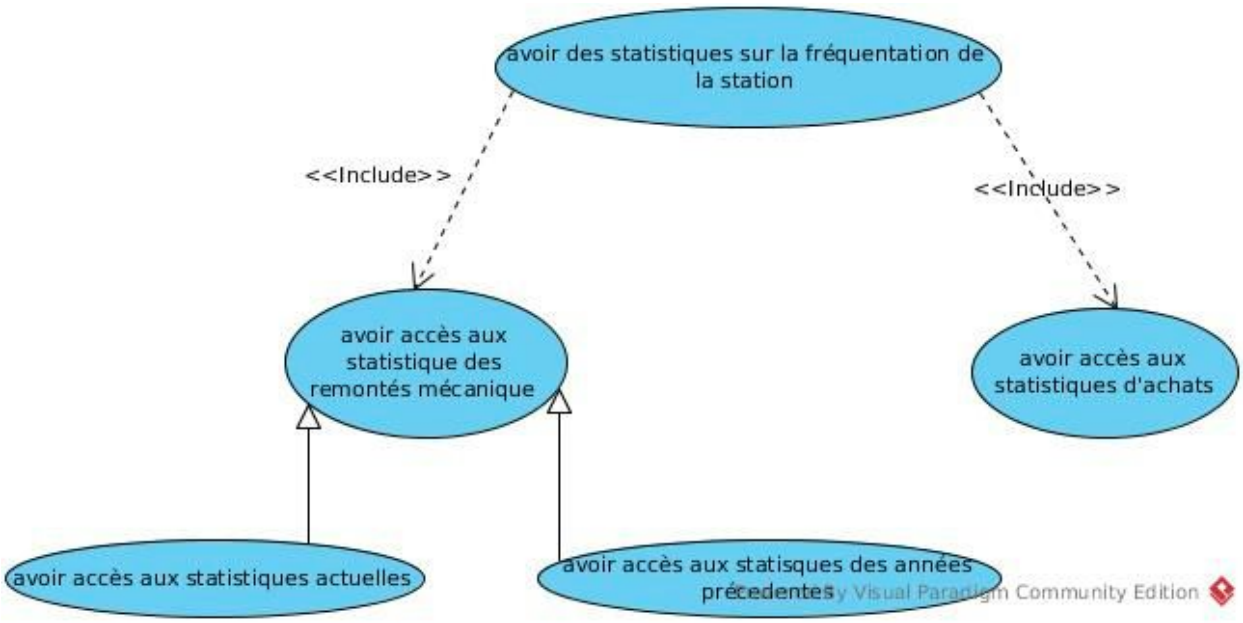


ci desssus le diagramme UC d'I*SOLA 3000-system* contenant les grandes fonctionnalités du systèmes ainsi que les différents acteurs hiérarchisés, ce dernier nous permet d'avoir une vision globale sur *ISOLA3000-system*
ci dessous on développera à la suite les différents use cases en sous use cases.

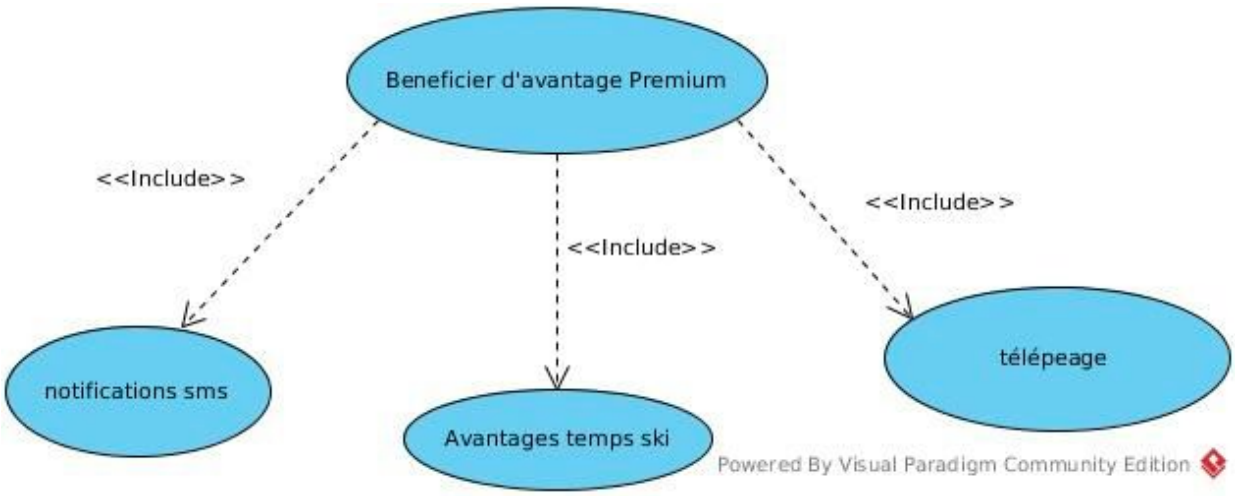


un utilisateur souhaite acheter un forfait, il peut le faire de deux manières :

- sur internet : si c'est un nouveau client il crée un nouveau profil, si non il se connecte en utilisant sont nom d'utilisateur et son mot de passe, dans son profil il choisis la section achat en ligne, il choisis un forfait, le nombre, et peux éventuellement aussi acheter un pack, ensuite il enregistre sa carte CIME si il en a une, si il n'en a pas elle lui est facturé au moment du payement, puis il paye sont forfait avec une carte de crédit.
- en caisse : le client demande en caisse les forfaits qu'il veux acheter, il communique les informations à la caissière afin qu'elle puisse créer le profil, puis le client payes le prix des forfait demandés et de la carte CIME.

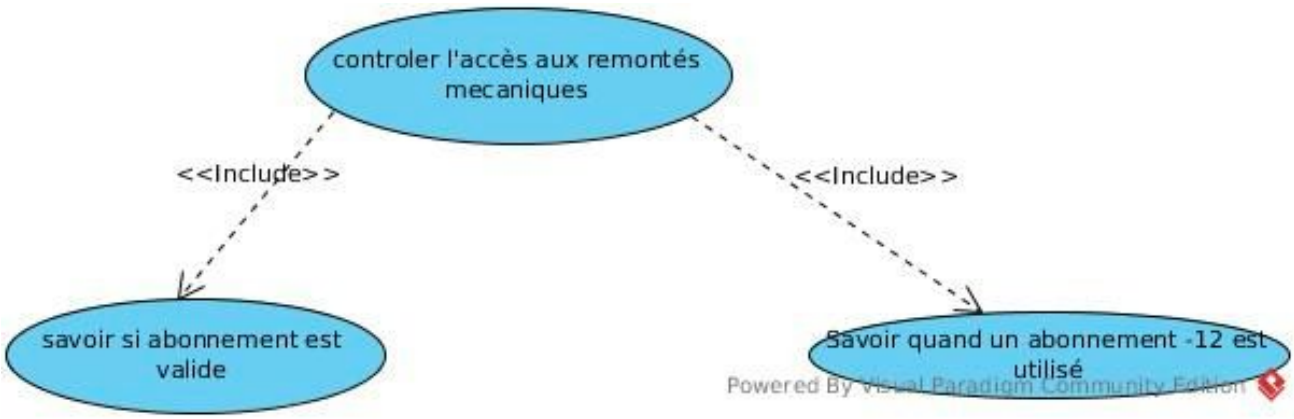


- un commerçant abonné peut consulter les statistiques en se connectant sur le profil dédié a son commerce, il peut avoir les statistiques partielles : l'âge moyen de la clientèle, l'âge de la clientèle qui consomme le plus, les lieux les plus fréquenté l'année dernière ou cette année.
- le poste de contrôle à accès à l'ensemble des statistiques du domaine en détails, taux de remplissage moyens d'une remontée, les zones les plus fréquentés l'année dernière ou cette année, les zones saturés afin de rediriger les skieurs.

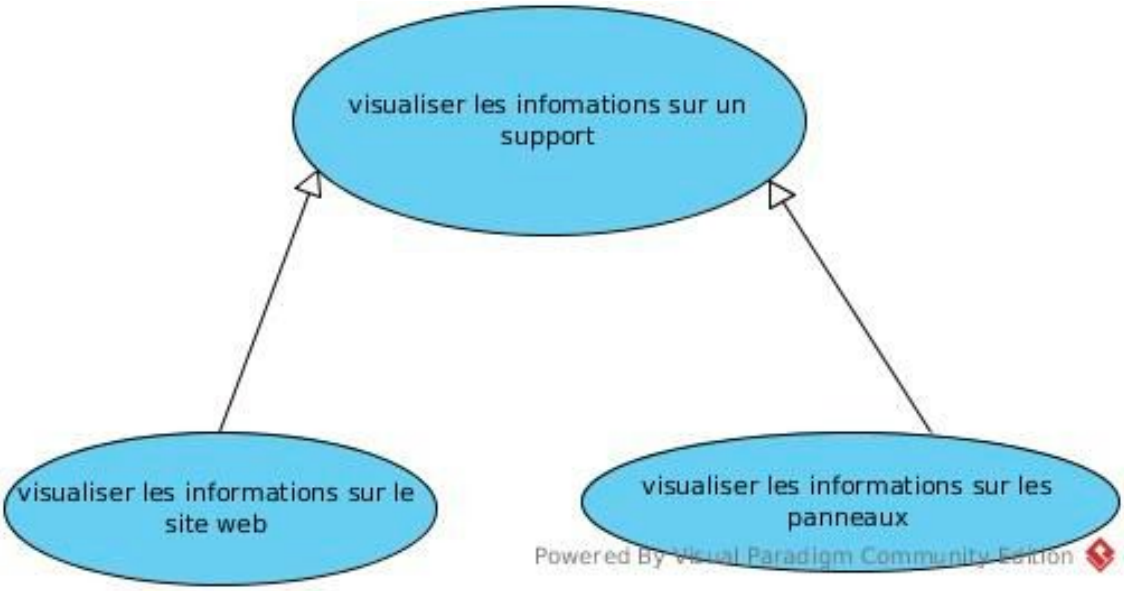


les Premium quand a eux peuvent :

- être notifier par sms.
- avoir des avantages temps de ski.
- télépeage : leur consommation est débiter automatiquement.

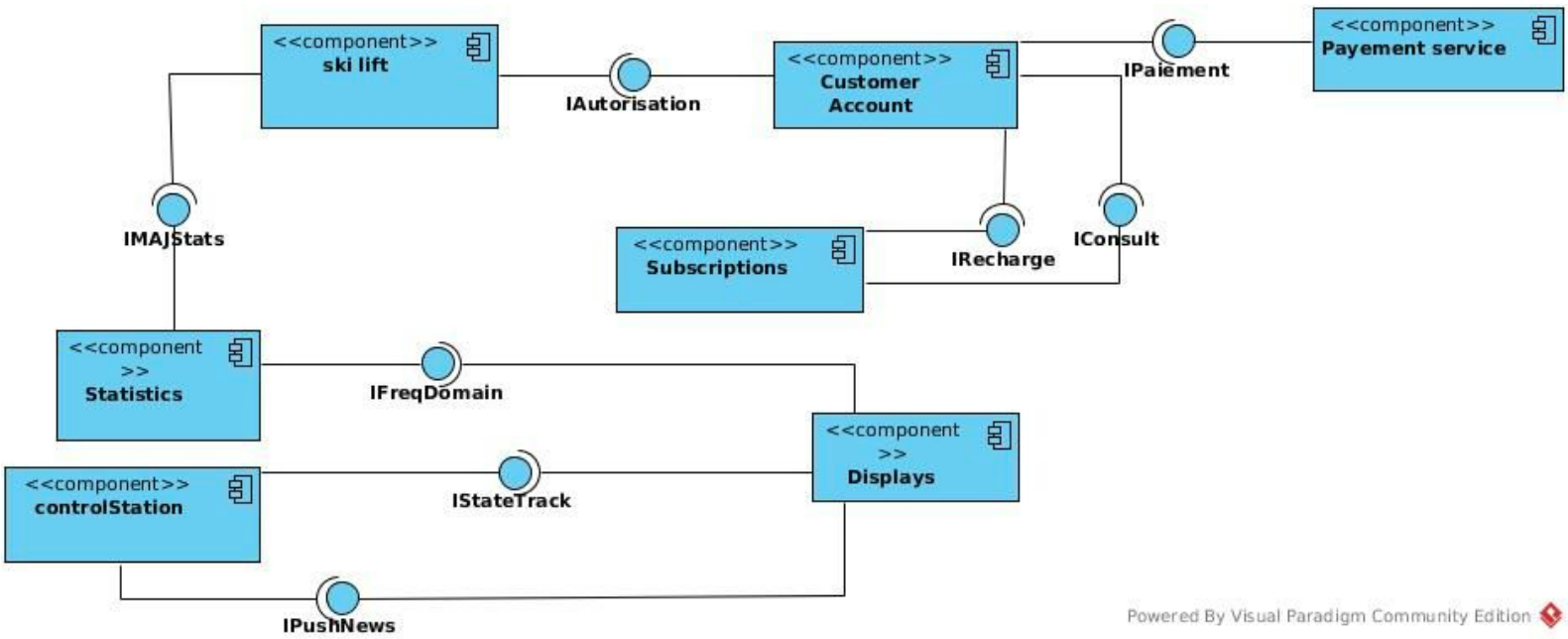


ISOLA3000-system permet aux perchistes de savoir si un forfait -12 ans est utilisé, et refuser un abonnement non valide et notifier le perchiste.



un abonné doit pouvoir visualiser sur les différents panneaux la fréquentation des différents zone afficher automatiquement, ainsi que les messages envoyé depuis le centre de contrôle (les messages étant afficher du plus prioritaire au moins prioritaire).

1.2 - Diagramme de composant



on remarquera que le composant *SkiLift* supposé connecté en utilisant des satellites vu le nombre faible ces remontés mécaniques, ce qui permettrait de contrôler l'accès a ces zones reculés prisés par les professionnels et les amateurs de sensation forte (et qui pourrait constituer donc une clientèle assez importante en terme de consommation d'abonnements)

voici maintenant en détails les différents contrats fournis par les différents composants

IMAJStats

- \+ pullStats () : List<SkiLiftStats>
- \+ pullStatsOf(Zone z) : List<SkiLiftStats>
  z : *la zone où on veux récupérer les statistiques*

SkiLiftSats : *contient toutes les statistiques d'une remontée mécanique*

IAutorisation

- \+ getAge (IDCime) : int

  IDCime *: Identifiant de la carte CIME donc les autorisations sont à vérifier*
  int *: renvois l'âge de du propriétaire de la carte*

+ getAutorisations(IDCime) : Liste<Zone>

Zone = *Zone, Remontée ou autre passage dont les forfaits permettent un accès réservé.*

List<Zone> *: la liste des zones que le forfait lié à la carte CIME en paramètre permet d'accéder au moment de la requête.*

IPayment

+ pay(value : float) : Voucher
  - *value : valeur en euros du montant que l'usagé va payer*
  - *Voucher : la méthode retourne un objet qui représente le reçu du payement avec tout les information concernant la transaction*

IRecharge

+ recharge(ca : CustomerAccount, type : TypeSub) : void
  - ca : *compte utilisateur à créditer*
  - type : *type de forfait à ajouter à l'utilisateur*

IConsult

+ getPermissionsZones() : List<Zone>
  - *List<Zone> : représente la liste des zones du domaine qui sont soumis a des forfaits*

+ getPersmissionTimes() : TimeStamp
  - *TimeStamp : les différentes durées proposé par les forfaits*

+ getTypeSubs() : List<TypeSub>
  - List<TypeSub> : la liste des différents types de forfaits proposé par la stations

IFreq

+getAllTrafficInfos() : Map<Zone, double>
  - Map<Zone, double> : *couple de donnée contenant la zone et la valeur représentant une moyenne du trafic*

+geTrafficInfos(zone : Zone) : double
  - Zone : *zone où calculer la fréquentation moyenne*
  - double : traffic moyen de la zone

IStateTrack

+ recupererRecommendations() : Queue<Message>

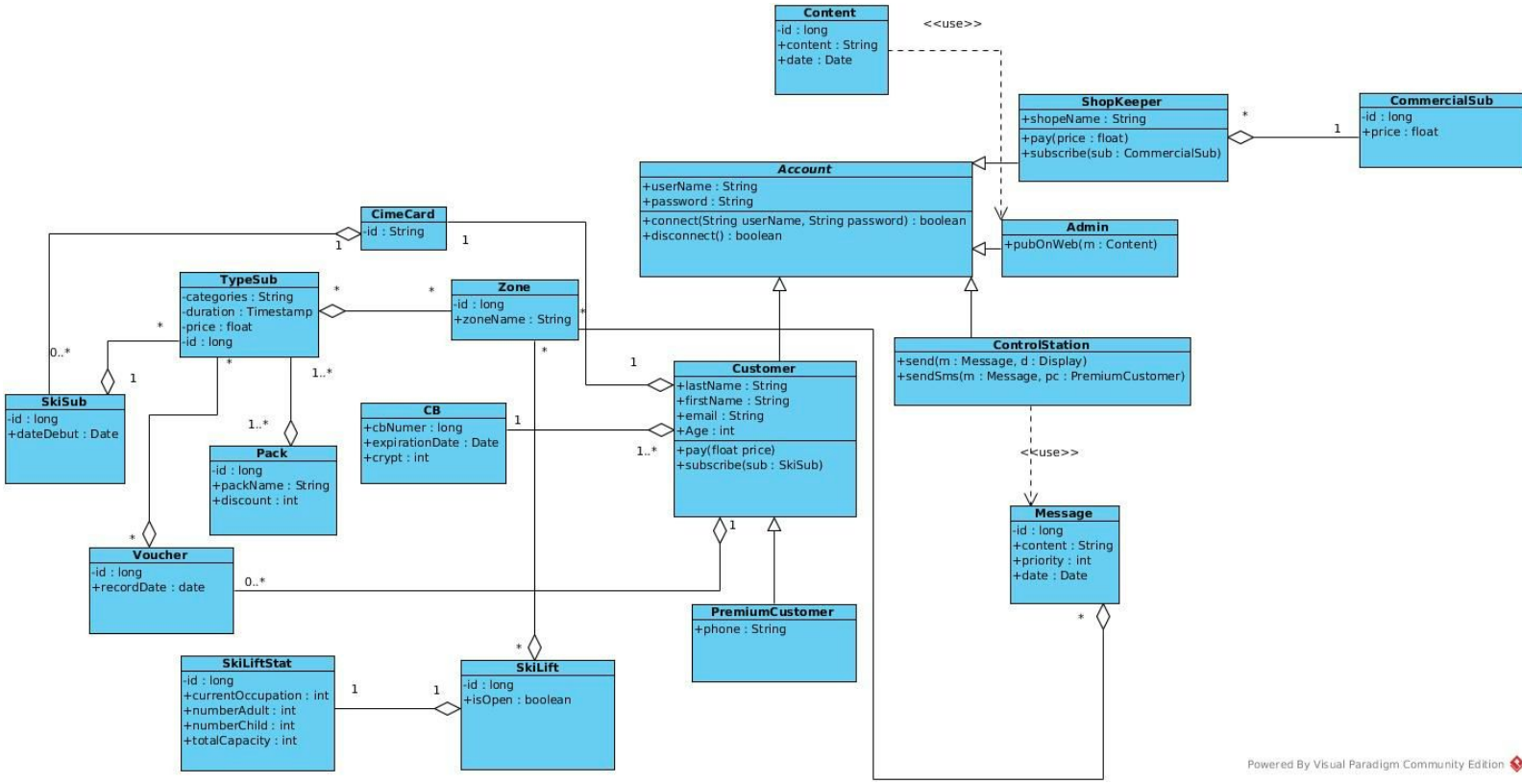
- Queue<Message> : message ordonnés suivant leur priorités, la priorité d'un message est définie par l'équipe du poste de contrôle

IPushNews

+ modifierMessageAffiché(message : Message) : void
    - message : *essentiellement le contenue du message mais aussi la date pour l'identifier*

II - Vue de développement

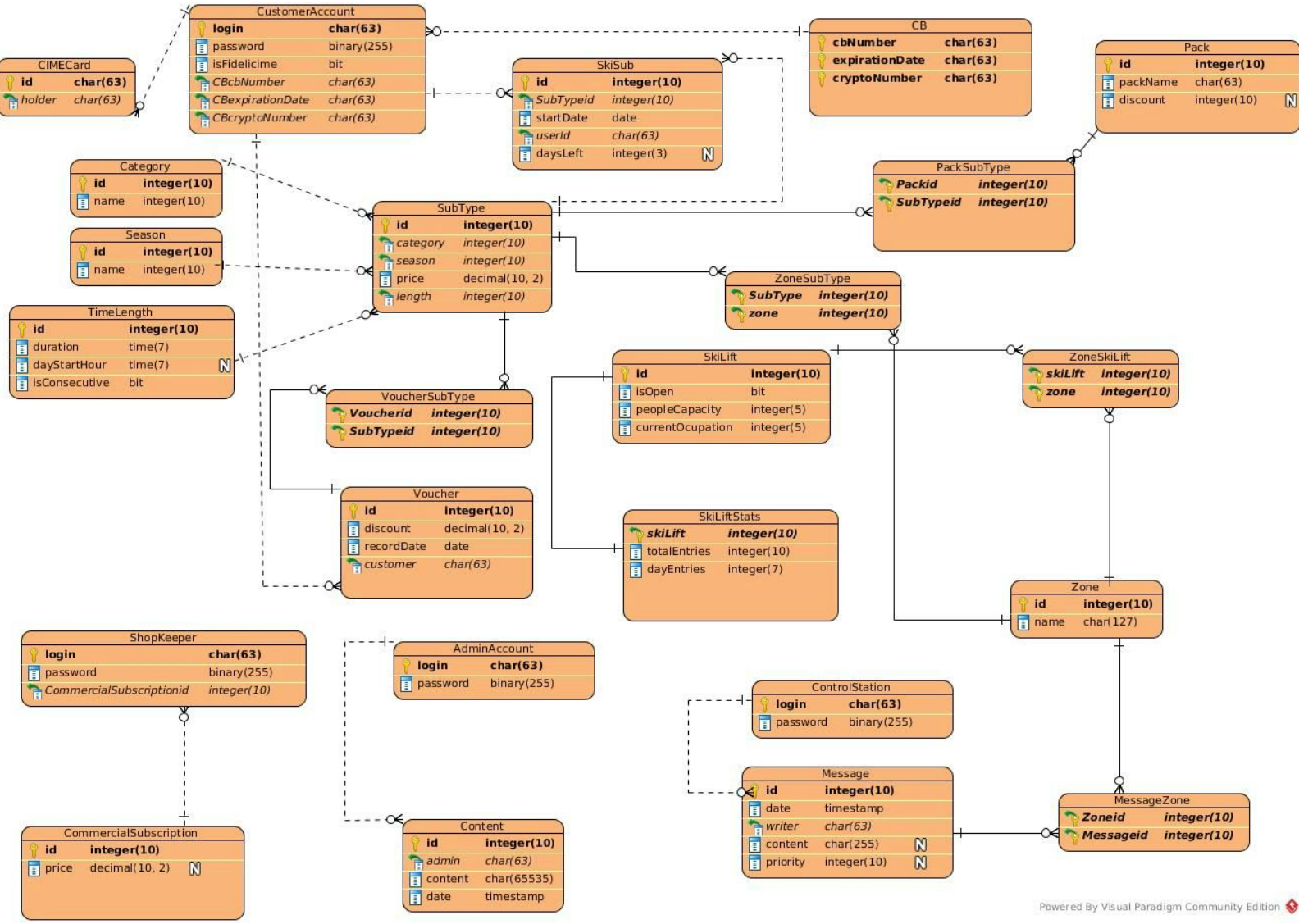
2.1 - Diagramme de classes des objets métiers



en ce qui concerne le diagramme de classes des objets métiers, les donnés des cartes bancaires et des cartes CIME sont modélisé car elle seront manipulé, les cartes "physique" ne font donc pas partie de ISOLA300-système, mais en revanche les donnés qu'elle contiennent oui.

2.2- Modèle relationnel de stockage

2.2.1 - Le Diagramme



2.2.2 - Justification des choix dans modèle relationnel

le système d'isola 3000 étant un système transactionnel il doit répondre à certains critères, le plus important étant RFIT (**R**apidité **F**iabilité **I**nfléxibilité **T**ransaction commandé)

pour assurer cette rapidité du système de compte, le choix c'est porté sur la mise en place du *concrete-class pattern* au niveau du modèle relationnel, ce qui nous évites de faire de multiples jointures, on peux penser que cela aurait comme effet de bord de nous faire faire des recherche sur les 3 tables (CustomerAccount - controlStation - Admin) pour savoir qui c'est connecté, mais ce sera implicitement l'utilisateur qui nous fourniras l'information de où chercher.
un des désavantage de la solution est que si on ajoutes a la classe compte un nouvel attribut alors il faudrait ajouter cet attribut dans les tables customer, controlStation et Admin, mais ce désavantage et un des moins désavantageux pour nous car au final la modélisation d'un compte ne change pas vraiment (nom d'utilisateur et mot de passe).
de plus le choix d'un type de compte est toujours déterministe (on ne peux pas avoir un compte qui soit client adminWeb en même temps, le problème de "*dans quelle table stocker l'utilisateur ?*" ne se pose donc pas).
le changement de statut d'un client ne peut s'opérer fréquemment que dans un sens (CustomerAccount -> prémium) puisque l'abonnement est annuel, donc les accès en écriture du bit qui indique si un client est premiums ou non sont diminués.

plusieurs tables d'associations sont présentent sur le modèle relationnel, cela viens des différentes relation N..M présentes sur le diagramme de classe, de plus l'information n'est pas dédoubler vu que les tables d'associations ne contiennent que des pointeurs logique vers les "vrais" informations contenues dans les autres tables.

3. Mapping objet - relationnel

2.3.1 - Explicitation de la couche de persistance

la couche de persistance a pour rôle de découpler la logique métier du moyen de stockage, les objets n'ont pas à savoir comment il sont stocker et la base de donnés n'a à savoir quels sont les objets manipuler, cette couche est composer de Finders, Gateways, Data Mappers, RecordSet (la mise en place de ces pattern a pour but d'éviter une *upward dependency)* et de UnitOfWork.

- toutes les classes dites concrètes aurons besoin d'un finder et d'une Gateway découplés (la classe Account étant une classe abstraite elle n'aura pas besoin de Finder et de Gateways)
- Mise en place d'une UnitOfWork pour les forfait ce qui permet une traçabilité de ce qui se passe lors des différentes transaction (par exemple les achats de forfaits)
- l'utilisation d'un RecordSet qui s'interface avec ResultSet retourner par ADO.NET ou java.sql donc notre Gateway retourn une interface RecordSet et le Data Mapper de la classe se charge de faire le mapping et de retourner l'objet

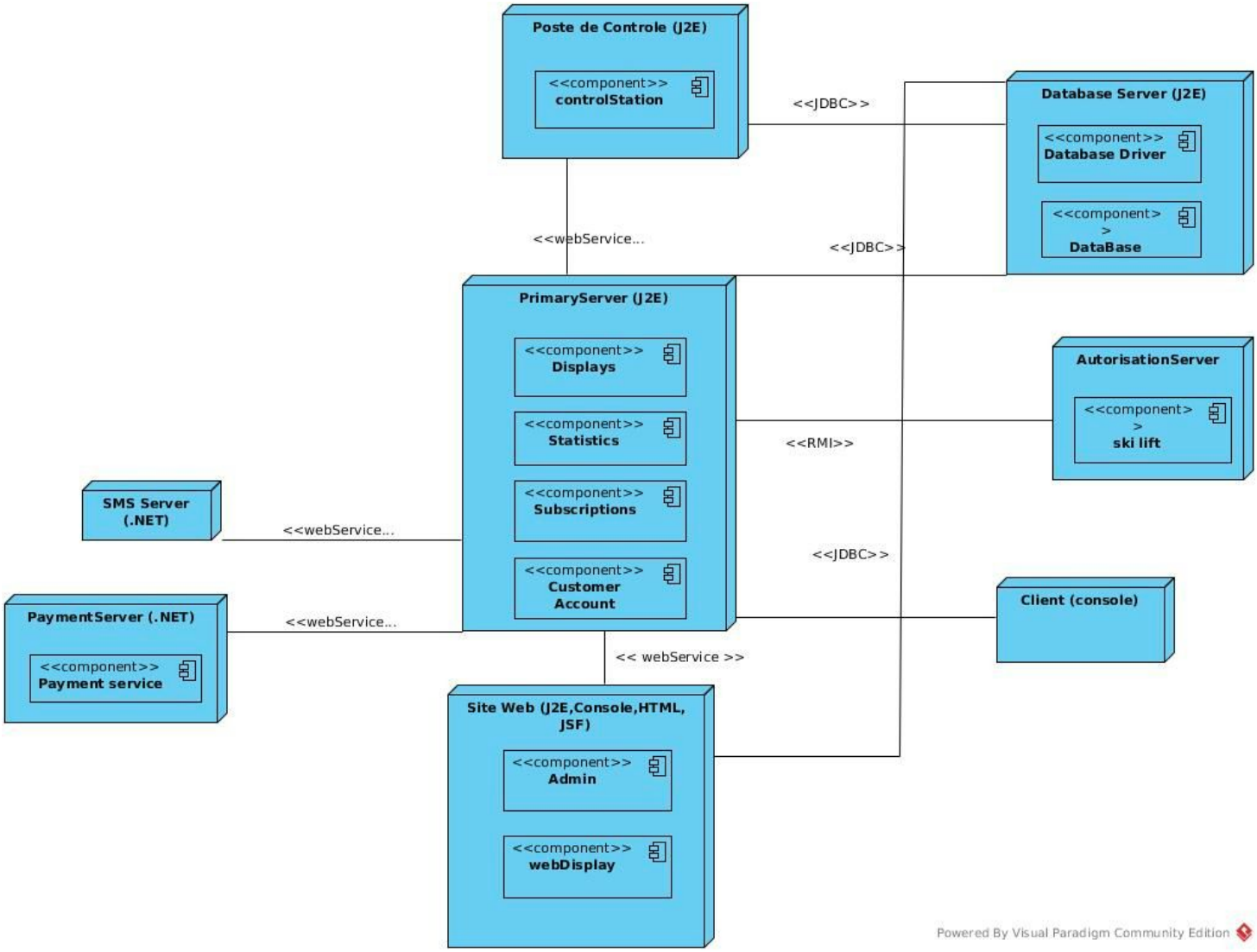
la question qui se pose et dans quelle couche se situe les Finders, Gateways, Data Mappers, RecordSet ? toutes se situent dans la couche de persistance, sauf les Mappers qui sont dans la couche métier ce qui nous permet de ne pas avoir de *upward dependency* entre nos couches.

en ce qui concerne la mise en place de cache, la mise en place du pattern *identity map* semble approprié, cependant que mettre dans le cache ? les les différents types de forfait et les différentes zones du domaine semble être une bonne idée du fait que ces deux derniers changent assez peux, le risque d'inconsistance pour ce type de donnés et donc assez réduit.

en ce qui concerne le *lazy loading* pattern pour cette conception il semble ne pas être assez approprié dans la mesure où ce dernier nécessite un chargement partiel du graphe de dépendance entre les objets, ce qui augmente le nombre d'appels à la couche de persistance et qui coûte assez chére (c.f la règle des 10).

III - Vue de déploiement

3.1 - Diagramme de déploiement



en ce qui concerne le déploiement il se fait de manière à avoir le serveur primaire, la gestion du site web et le poste de contrôle en JEE
les services tiers tel que le service paiement ou de notification sont fait en NET et déployer sur des serveurs distants (certaines options de déploiement sont encore à l'étude a ce stade pour achever un scaling horizontale)

Conclusion

ce rapport d’architecture s’efforce de plus donner une vision de l’architecture qu’une roadmap pour la réaliser de bout en bout, l’architecture est vouée inévitablement à bouger suivant les besoins du client et les contraintes technique, donc on vise plutôt l'adaptabilité que la complétude de l’architecture.