POLYTECH NICE-SOPHIA

INTRODUCTION À L'ARCHITECTURE LOGICIELLE

Dimanche 21 février 2016
Enseignant responsable : Sébastien MOSSER

Groupe AA
Nicolas Hory
Lucas Martinez
Lucas Soumille

Sommaire

Introduction

Avec l'essor des smartphones et des technologies toujours plus innovantes, la société d'exploitation des remontées mécaniques Isola 3000 a décidé de moderniser son système d'information. Pour cela, elle a lancé un appel d'offre indiquant dans les grandes lignes, les nouvelles fonctionnalités attendues.

Tout au long de ce rapport, nous allons présenter l'architecture de notre plateforme. Nous commencerons par spécifier les différents scénarios qui représentent les principales fonctionnalités. Ensuite, nous verrons les liens entre les composants, classes et relations mis en jeu dans l'application. Enfin, nous proposerons une perspective de déploiement.

Vue fonctionnelle

I. Scénarios

En lisant l'appel d'offre, nous avons déterminé un ensemble de scénario. Chaque scénario correspond à une fonctionnalité de l'application.

- **Scénario n°1 : Vendre des cartes cime**

  - Achat d'une carte à la caisse
  - Réserver une carte sur le site web
    - Choix de la caisse pour la retirer
  - Générer un identifiant de carte unique

- **Scénario n°2 : Vendre des forfaits**

  - Vendre un forfait à la caisse
  - Vendre un forfait sur le site web
  - Télécharger le forfait sur le carte cime
  - Proposer des forfaits spéciaux
    - Réduction lors de l'achat
    - Service "fideli'cimes"
  - Rembourser des forfaits à cause des conditions météo

- **Scénario n°3 : Contrôler l'accès aux remontées mécaniques**

  - Vérifier la validité du forfait lors du passage sur le portique
    - Activation du forfait
  - Communiquer avec les remontées mécaniques isolées
  - Déclencher une alarme lors du passage d'un forfait -12 ans

- **Scénario n°4 : Collecter les statistiques**

  - Collecter des informations concernant les ventes de forfait
  - Récupérer les charges des remontées mécaniques en temps réel
  - Assurer la persistance sur plusieurs années

- **Scénario n°5 : Communiquer les statistiques**

  - Proposer des recommandations aux clients sur le site web
  - Envoyer des informations aux commerçants abonnés
  - Créer des bulletins des pistes qui seront publiés sur le site
  - Informer les pisteurs pour la mise à jour des panneaux “fixes”
  - Mise à jour automatique des panneaux “dynamiques”

- **Scénario n°6 : Informer les usagers**

  - Récupérer les prévisions météorologiques
  - Notifier les clients grâce à des SMS
  - Définir des sous-ensembles d’usagers
    - Créer des groupes d’usagers en fonction de critères précis (âge, type de forfait)
    - Envoyer des informations seulement à un groupe

II. Diagrammes de cas d'utilisation

Le diagramme de cas d'utilisation de haut niveau ci-dessous regroupe les différentes fonctionnalités que nous avons pu mettre en évidence dans les scénarios, tout en spécifiant les acteurs qui interagissent avec ceux-ci. Il permet d'avoir une vue générale de l'aspect fonctionnel de notre application.

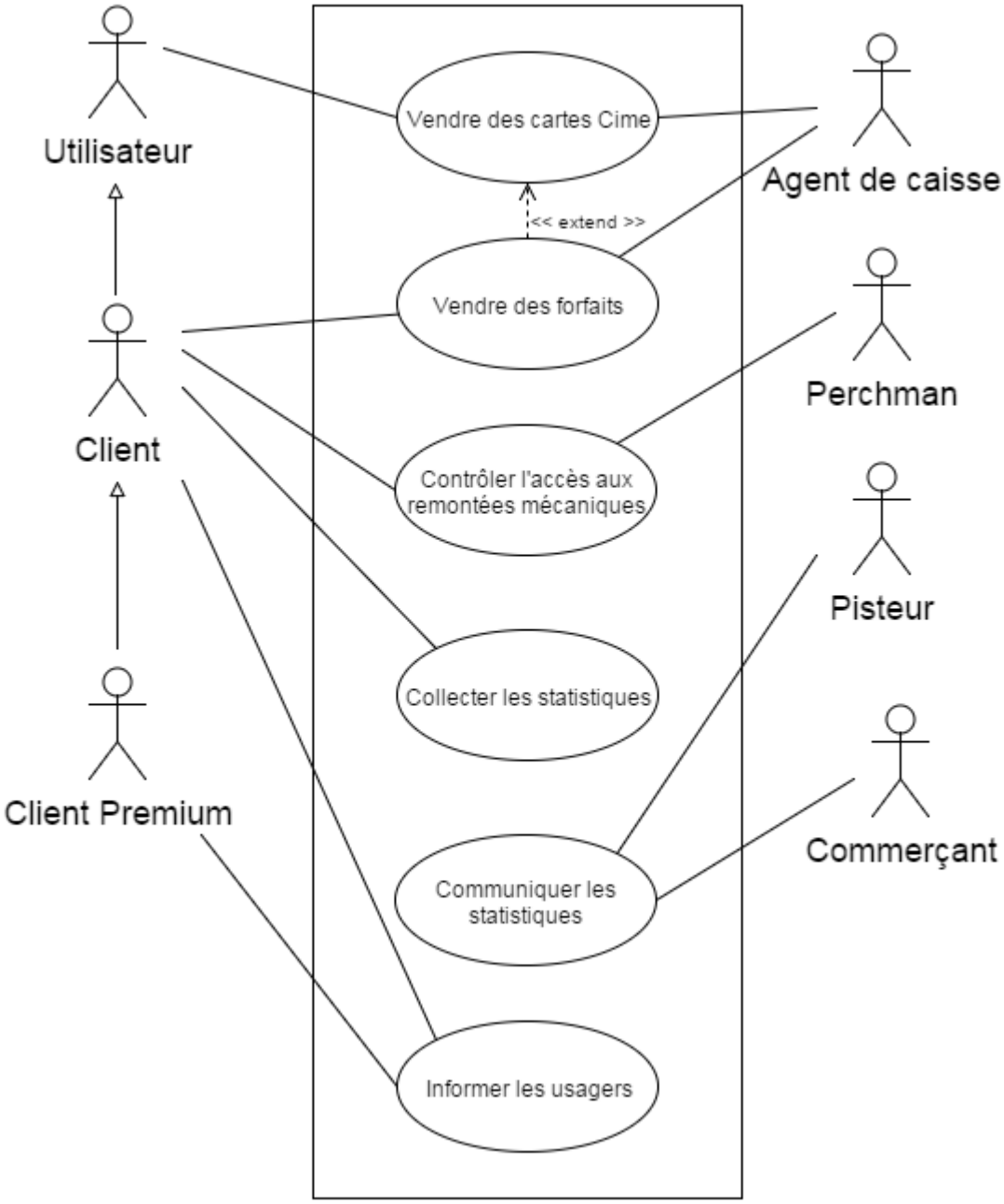

Figure 1: Diagramme de haut niveau

Nous avons déterminé sept types d'acteurs que l'on peut classer en deux parties. Une première regroupe les membres de la société à laquelle nous pouvons rajouter le commerçant. La deuxième est une hiérarchie d'utilisateur qui évoluera en fonction des services souscrits.

Cas d'utilisation : Vendre des cartes Cimes

Afin de pouvoir "utiliser" notre système d'information, c'est à dire acheter des forfaits, passer les portiques des remontées ou encore s'abonner à des services premium, l'utilisateur doit posséder une carte Cime. Sans celle-ci, il est impossible d'accéder aux autres fonctionnalités. Dès qu'un utilisateur possède cette carte il est alors considéré comme client.

Voici le diagramme de cas d'utilisation (abrégé UC pour *Use Case* par la suite) associé à la description ci-dessus.

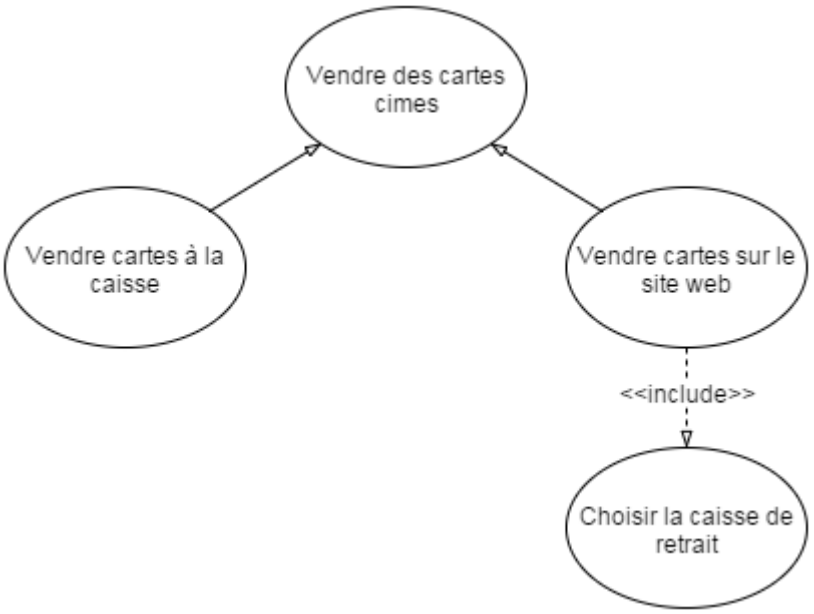

Figure 2: Diagramme UC : Vendre des cartes cimes

Cas d'utilisation : Vendre des forfaits

Dès qu'un utilisateur possède la carte, il peut acheter un forfait. Tout comme la carte, il peut les acheter directement sur place à la station ou recharger sa carte sur le site web s'il possède un compte. Le client peut aussi utiliser sa carte comme un télépéage s'il paye la cotisation nécessaire, c'est-à-dire en souscrivant au service fidéli'cime.

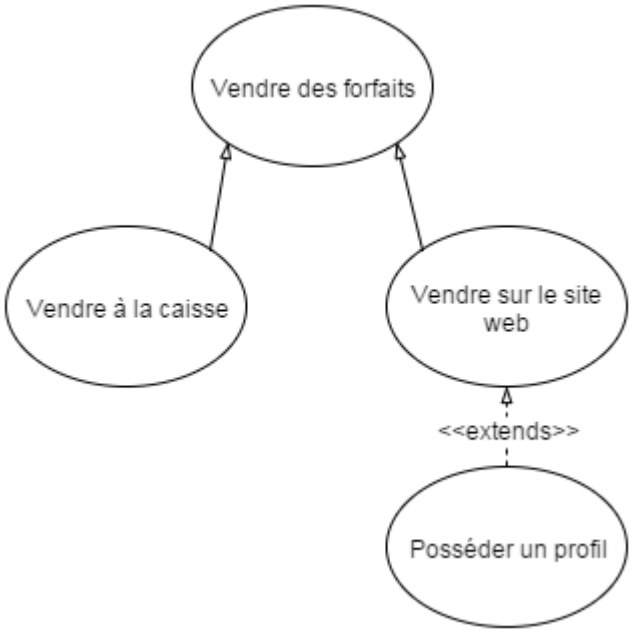

Figure 3: Diagramme UC : Vendre des forfaits

Cas d'utilisation : Contrôler l'accès aux remontées mécaniques

Le client avec une carte chargée d'un forfait pourra passer les portiques afin d'accéder aux remontées mécaniques. A chaque passage une vérification de la validité du forfait sera faite. Si l'utilisateur possède un forfait valide et qu'il passe pour la première fois, son forfait sera activé. Si le forfait vérifié par le portique est un forfait jeune (moins de 12 ans), une alarme sera déclenchée.

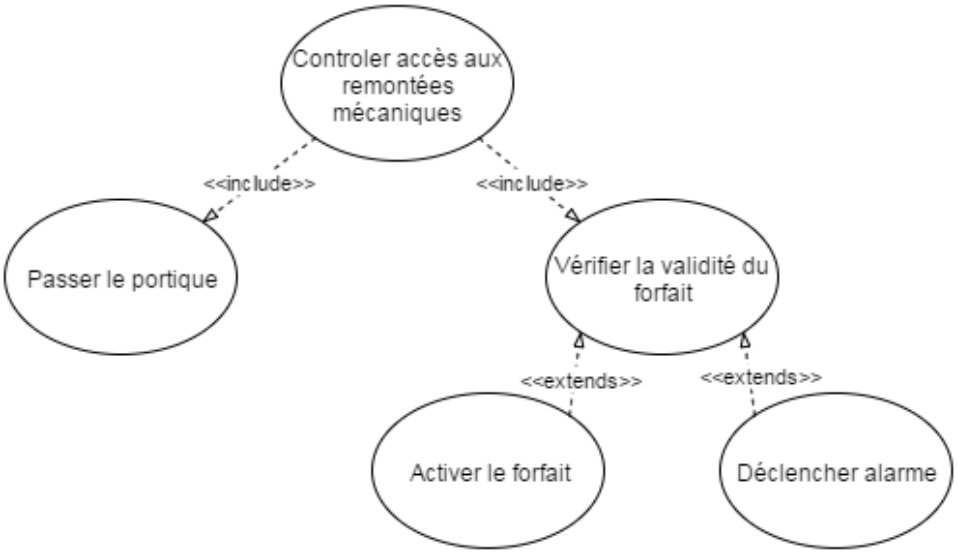

Figure 4: Diagramme UC : Contrôler l'accès aux remontées mécaniques

Cas d'utilisation : Collecter les statistiques

L'exploitation des statistiques est une partie importante de l'application. Il faut tout d'abord collecter les informations. Ici, nous souhaitons recueillir les statistiques sur les ventes de forfait et connaître les charges sur les différentes remontées des stations en temps réel.

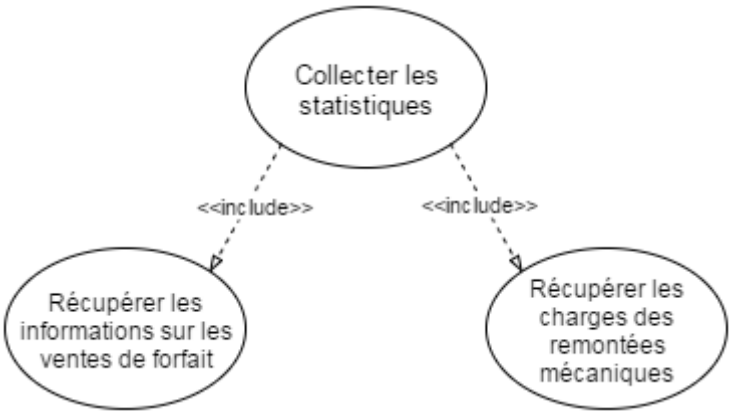

Figure 5: Diagramme UC : Collecter les statistiques

Cas d'utilisation : Communiquer les statistiques

Une fois que nous possédons ces statistiques, il faut les traiter afin de pouvoir les diffuser. La diffusion concerne des personnes physiques telles que le pisteur pour la mise à jour des panneaux manuels ou les commerçants abonnés. Aussi, de nouvelles statistiques impliquent la mise à jour de bulletin sur le site web et le rafraîchissement des panneaux dynamiques.

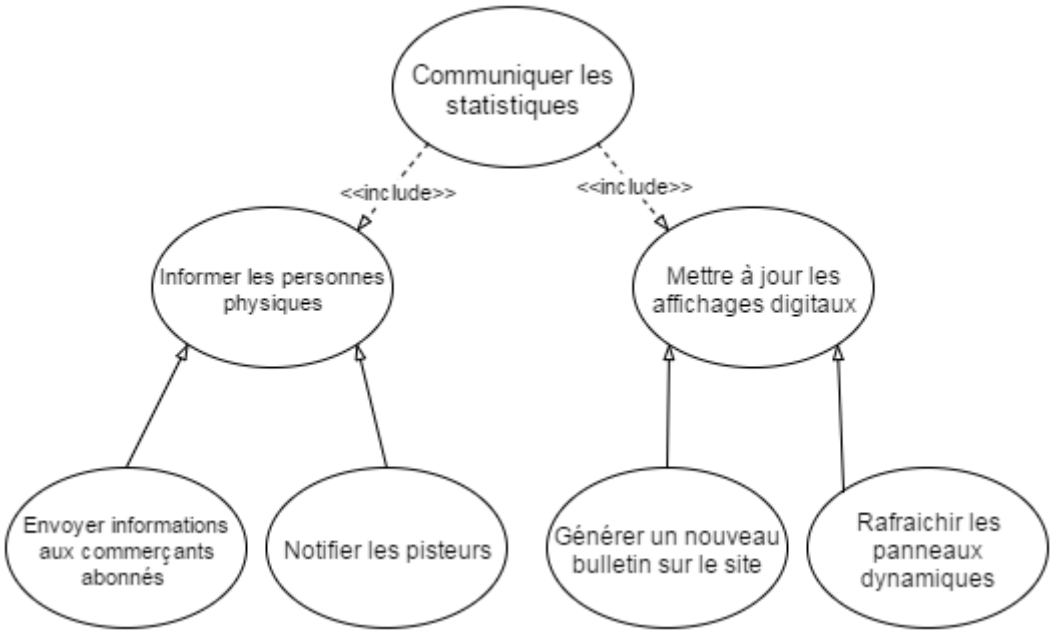

Figure 6: Diagramme UC : Communiquer les statistiques

Cas d'utilisation : Informer les usagers

Certains clients peuvent obtenir le statut de client premium. Ils seront alors notifiés de l'état d'enneigement de la station et des dernières promotions. Afin de remplir la station, la société souhaite notifier des clients d'offres spéciales lors des périodes creuses.

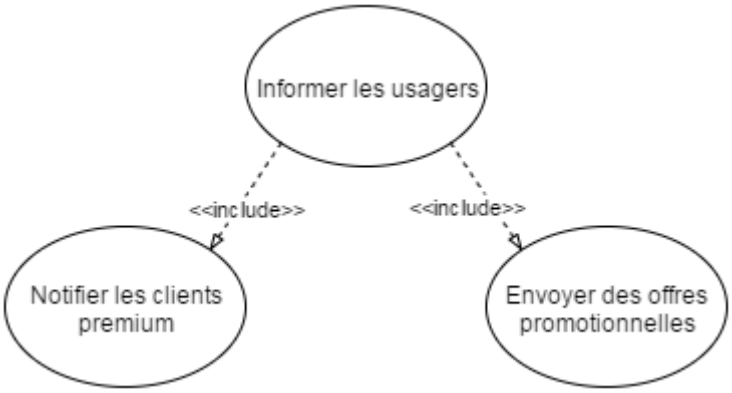

Figure 7: Diagramme UC : Informer les usagers

III. Diagramme de composants

Le diagramme ci-dessous représente l'ensemble des composants de l'application Isola 3000 en spécifiant les différentes fonctions implémentées afin qu'ils puissent communiquer.

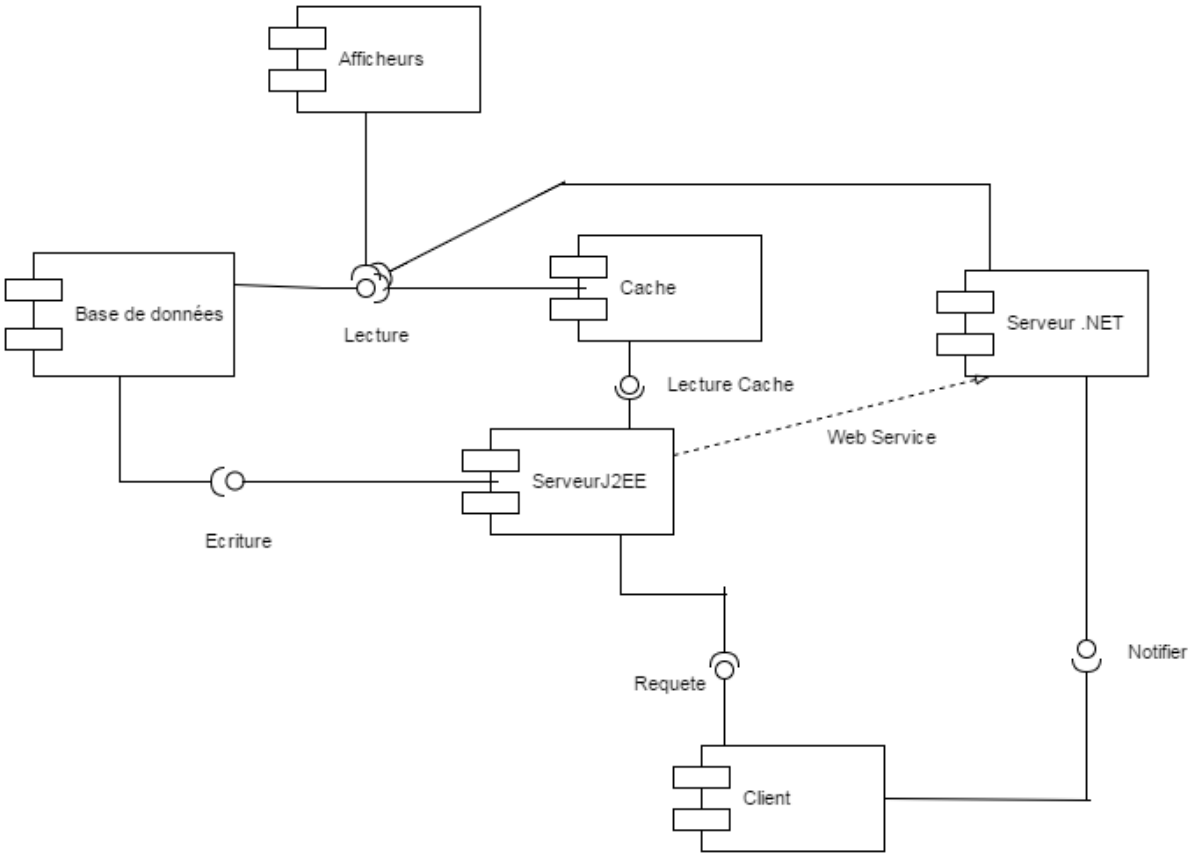

Figure 8: Diagramme de composants

L'application est composée de cinq composants. Le client peut faire des requêtes au serveur à partir de la caisse, sur le site web ou encore lors de son passage aux portiques. Le serveur assure la persistance des informations recueillies en les stockant dans la base de données. De plus, il communique avec le serveur .NET afin que celui-ci réalise certaines fonctionnalités annexes. Les afficheurs et le serveur .NET exécute des requêtes de lecture sur la base afin de se mettre à jour.
Nous avons pensé qu'il était judicieux de mettre un cache entre la base de données et le serveur J2EE afin d'accélérer les requêtes de vérification de forfait lors des passages aux portiques d'accès, sachant que les mêmes requêtes sont répétées à intervalle de temps assez court (après chaque descente pour un skieur).

Nous allons maintenant spécifier les différentes méthodes de chaque interface.

Requete

Les composants “serveurJ2EE” et “client” interagissent lorsque le client souhaite acheter un forfait pour le mettre sur sa carte ou lors de son passage au portique d’accès.
L’interface possède les méthodes suivantes :

- addForfait(Forfait forfait, String numCard)
- checkForfait(String numCard)
- createCard(String numCard)

Pour ces trois méthodes, le retour est une String qui contient le message de succès ou d’erreur de l’exécution de ces fonctions.

Ecriture

Le composant “serveurJ2EE” met à jour ou ajoute des informations dans la base de données en fonction des requêtes des clients.

- saveCard(String numCard)
- saveProfile(String nom, String numTel, String numCard)
- addForfaitOnCard(Forfait forfait, String numCard)
- activeForfait(Forfait forfait, String numCard)
- removeForfaitForCard(Forfait forfait, String numCard)

Lecture

L’interface lecture regroupe l’ensemble des fonctions qui permettent d’obtenir des données enregistrées dans la base. Celle-ci est utilisée par les composants “afficheurs”, “serveur .NET” et “cache”. Les méthodes de cette interface donnent les droits de lecture.

- getForfaitForCard(String numCard) / retourne une liste de forfait
- getActiveForfaitForCard(String numCard) / retourne une liste de forfait
- getSkiLiftInfo() / retourne une map avec le nom de la remontée et sa charge
- getProfiles / retourne une liste de profils
- getNbPeopleForStation(String s) / retourne le nombre de skieur sur la station
- getForecast() / retourne une map avec pour chaque jour la prévison météo (String)

LectureCache

Cette interface permet au composant “serveur J2EE” d’accéder aux informations contenues dans la base au travers du cache. On y retrouve deux méthodes de l’interface lecture

- getForfaitForCard(String numCard) / retourne une liste de forfait
- getActiveForfaitForCard(String numCard) / retourne une liste de forfait

Notifier

Le composant “serveur .NET” possède les fonctions relatives à la notification des usagers. C’est ce composant qui envoie les sms du service d’alerte poudreuse.

- sendNotification(Utilisateur util, String textNotif)

Connexion entre les deux serveurs

Le serveur .NET et le serveur J2EE communiqueront grâce à des requêtes composées de messages sérialisés en JSON.
Ce message sera composé de trois champs obligatoires et d’un champ optionnel :

1. Le champ Profile qui regroupe les informations sur l’utilisateur qui souhaite obtenir un service
2. Le champ Type qui permet de savoir la méthode à appeler pour le serveur .NET
3. Le champ Parameters qui concatène (séparés par des “|”) les paramètres nécessaires à l’exécution de la fonction
4. Le champ optionnel Answer qui sera rempli par le serveur .NET lorsqu’il a fini d’exécuter le service demandé et qu’il envoie une réponse au serveur J2EE

Le diagramme ci-dessous (Figure 9) représente les sous composants de “serveurJ2EE”.

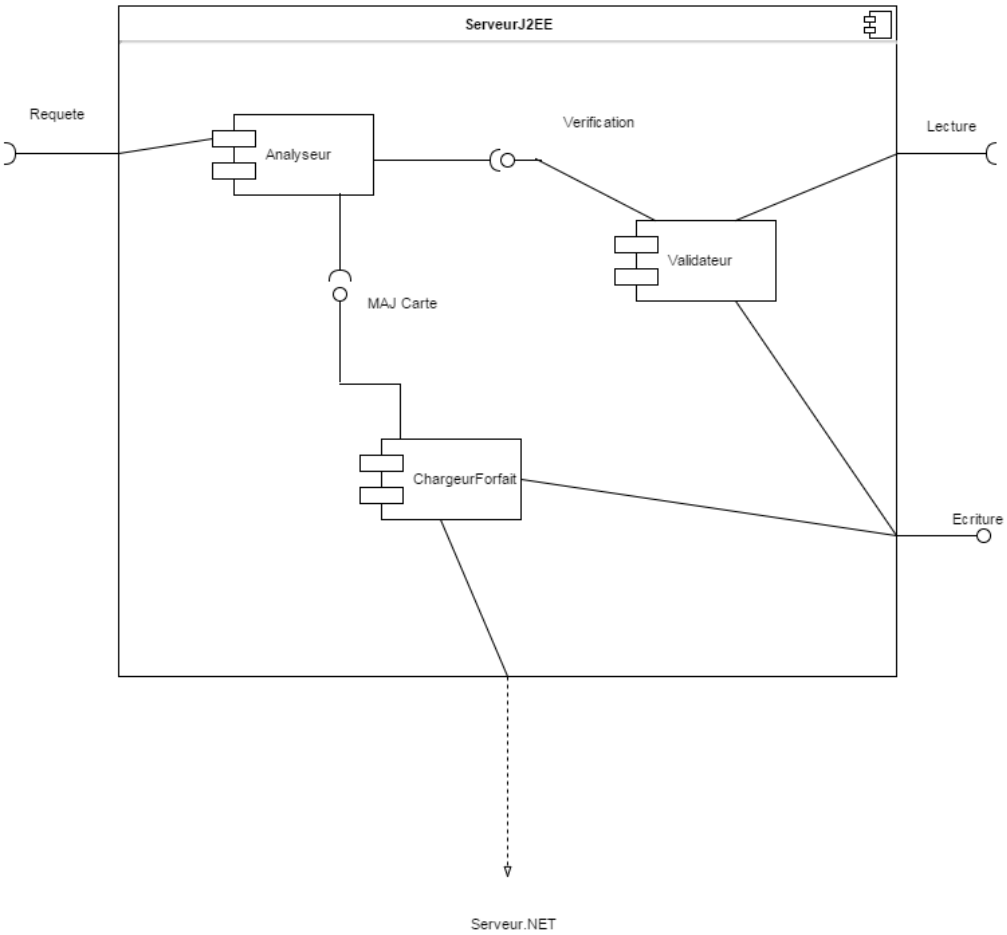

Figure 9: Diagramme de composant (Serveur J2EE)

Vérification

Le composant “analyseur” permet de répartir les requêtes du client vers le bon composant. Le composant “vérificateur” assure que le forfait présent sur la carte permet à l’utilisateur d’utiliser la remontée mécanique. L’interface propose ces méthodes de vérification :

- checkCard(String numCard) / retourne vrai si la carte existe
- checkForfaitForCard(String numCard) / retourne vrai si un forfait est valide sur la carte

Mise à jour carte

Le composant “achat” assure la disponibilité du forfait sur la carte après l’achat de celui-ci. Voici la fonction que propose l’interface :

- addForfait(Forfait forfait, String numCard)

Vue développement

I. Diagramme de classes

Pour modéliser notre application, nous avons défini les classes suivantes (Figure 10). Tout d’abord, une “*Person*” possède un identifiant privé unique en plus de ses informations (nom, prénom, âge). La classe dérivée “*ClientPremium*” rajoute des données supplémentaires afin que l’on puisse le contacter plus facilement.

La classe “*CarteCime*” possède l’identifiant unique de la carte en question que l’on générera à chaque nouvelle création de carte. Une carte est possédée par un utilisateur et peut contenir plusieurs forfaits.

Pour ce qui est des forfaits, nous les représentons à partir d’une hiérarchie qui a pour classe mère, une classe abstraite contenant l’attribut *isActive*. Dans les classes filles, nous pouvons retrouver tous les types de forfaits, que ce soit fidéli’cime ou des forfaits plus classiques. Nous avons fait une classe différente pour le forfait snowpark car il possède un traitement particulier à savoir un nombre limité d’utilisation de remontées. Une liste de remontées accessibles sera présente dans chaque forfait afin de faciliter la vérification de celui-ci lors de son passage au portique.

Les remontées sont modélisées selon leur moyen de communication avec le serveur, si elles sont connectées directement à Internet ou si elle communique par SMS pour les plus reculées. Chacune possède un nom et une localisation (station d’appartenance). Elles vérifieront les forfaits qui seront lus par leurs portiques.

Enfin, l'application sera composée de différents afficheurs qui permettent d'informer les utilisateurs. Chaque classe qui implémente l'interface "Display", dispose de la méthode permettant de lancer la récupération des données et une autre qui permet de mettre à jour l'affichage.

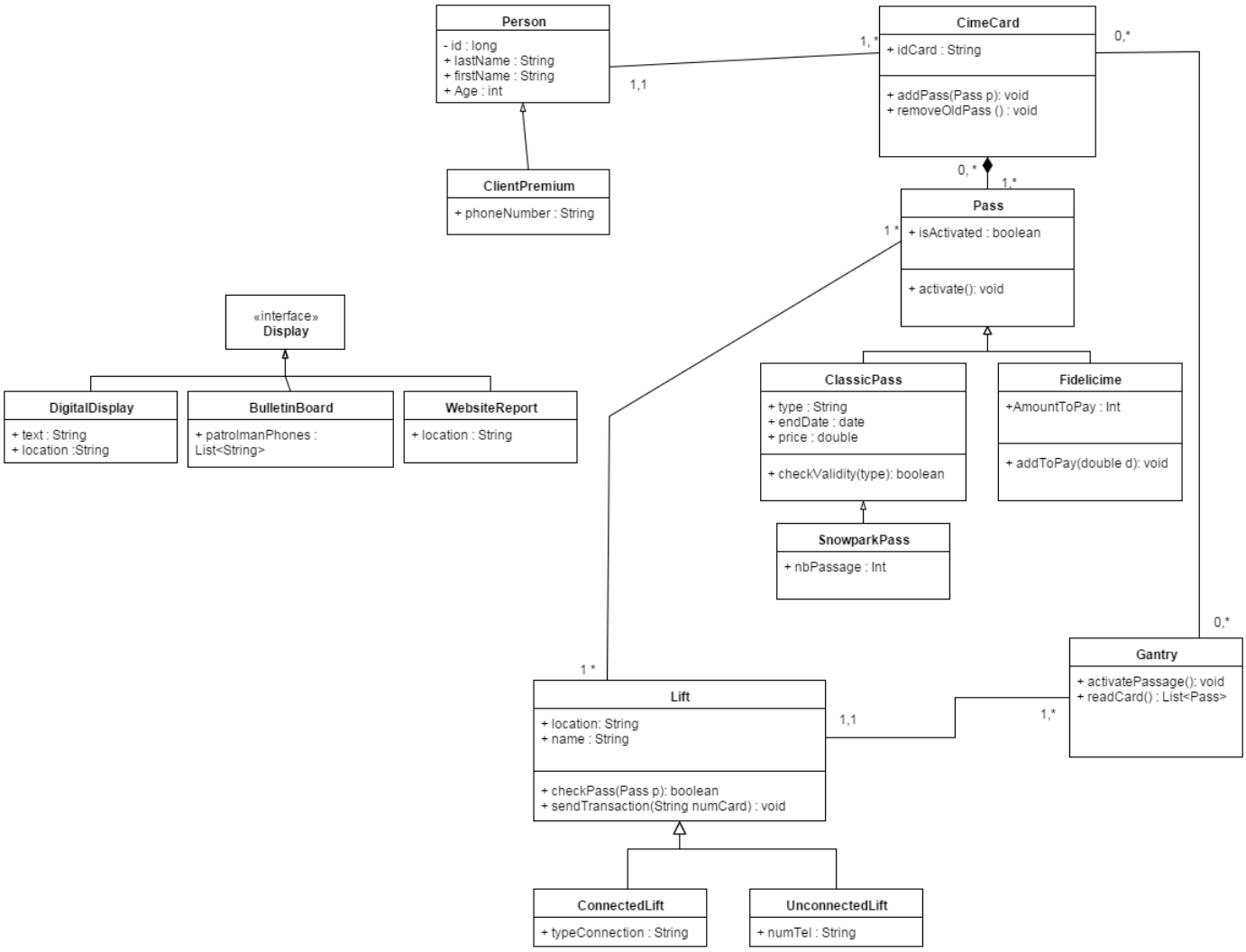

Figure 10: Diagramme de classes

II. Modèle relationnel

Une fois le diagramme de classes réalisé, il nous a fallu le transposer en un modèle relationnel permettant de modéliser les relations entre nos différents éléments. La première fonctionnalité étant la vente de cartes cime, le premier objectif était de représenter la relation entre un client et une carte. De ce fait, une fois les deux entités “Person” et “Card” créées, l’ajout d’une clé étrangère “idPerson” à l’entité “Card” a permis de représenter cette relation, puisqu’une carte ne peut appartenir qu’à une seule et même personne. Une relation entre ces deux entités aurait donc été inutile.

Pour représenter l’héritage de la classe “Person”, qui permettait de différencier les différentes personnes liées à notre application, nous avons décidé de rajouter des booleens isPremium (pour savoir si une personne est client Premium), et isCommerçant. Cela permet de savoir à quelle catégorie appartient une personne sans pour autant créer de nouvelle entité.

En ce qui concerne la vente de forfaits, qui fait également partie des fonctionnalités principales, le lien entre les cartes cime et les forfaits y étant associés est quant à lui représenté à l’aide de la relation “Contain”, qui a nécessité d’avoir les deux identifiants idCard et idPass puisque plusieurs forfaits peuvent être associés à une carte. Par ailleurs, nous avons décidé de distinguer les deux types de forfait, à savoir forfait classique et Fideli’Cime.

Pour ce qui est du contrôle de l’accès aux remontées mécaniques, qui se fait essentiellement à l’aide des portiques, il est représenté à travers la relation "Authorize” entre les entités “Card” et “Gantry”, qui, comme pour la relation “Contain”, nécessite les identifiants des deux entités puisque l’on peut associer une carte à plusieurs portiques, et plusieurs cartes à un même portique. De plus, cette relation possède l’identifiant du forfait concerné et une date, chose pouvant être utile par la suite pour les statistiques (par exemple récupérer le nombre de personnes passées à un portique en un jour).

Enfin, l’entité “Gantry” comporte un champ identifiant la remontée à laquelle un portique appartient en clé étrangère, afin de mettre en évidence la relation de composition. Nous avons également décidé de créer une entité “Lift” possédant un identifiant, un numéro de téléphone, un nom, une location et surtout un booleen indiquant si la remontée est connectée, paramètre entrant en compte pour la transmission de statistiques.

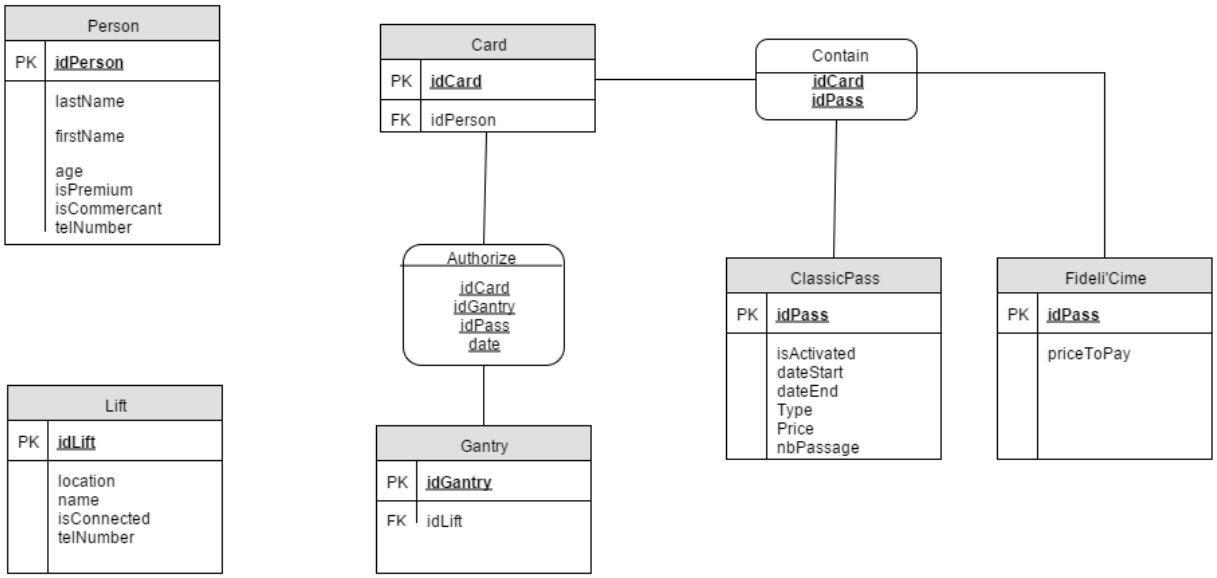

Figure 11: Modèle relationnel

III. Ajout de classes pour assurer la persistance

Nous avons utilisé le pattern "*Table Data*" avec une séparation des "*gateway*" qui permettent l'écriture en base et des "*finder*" qui permettent de récupérer les données. Nous l'avons choisi car il permet de séparer les fonctions qui permettent la gestion de la base, des fonctions métiers.

Nous avons décidé de séparer les classes d'écriture et de lecture (*gateway* et *finder*). Ainsi lors de l'accès à la base afin de récupérer des statistiques, nous utiliserons seulement un "*finder*" ce qui évitera les erreurs dûes à la modification des données.

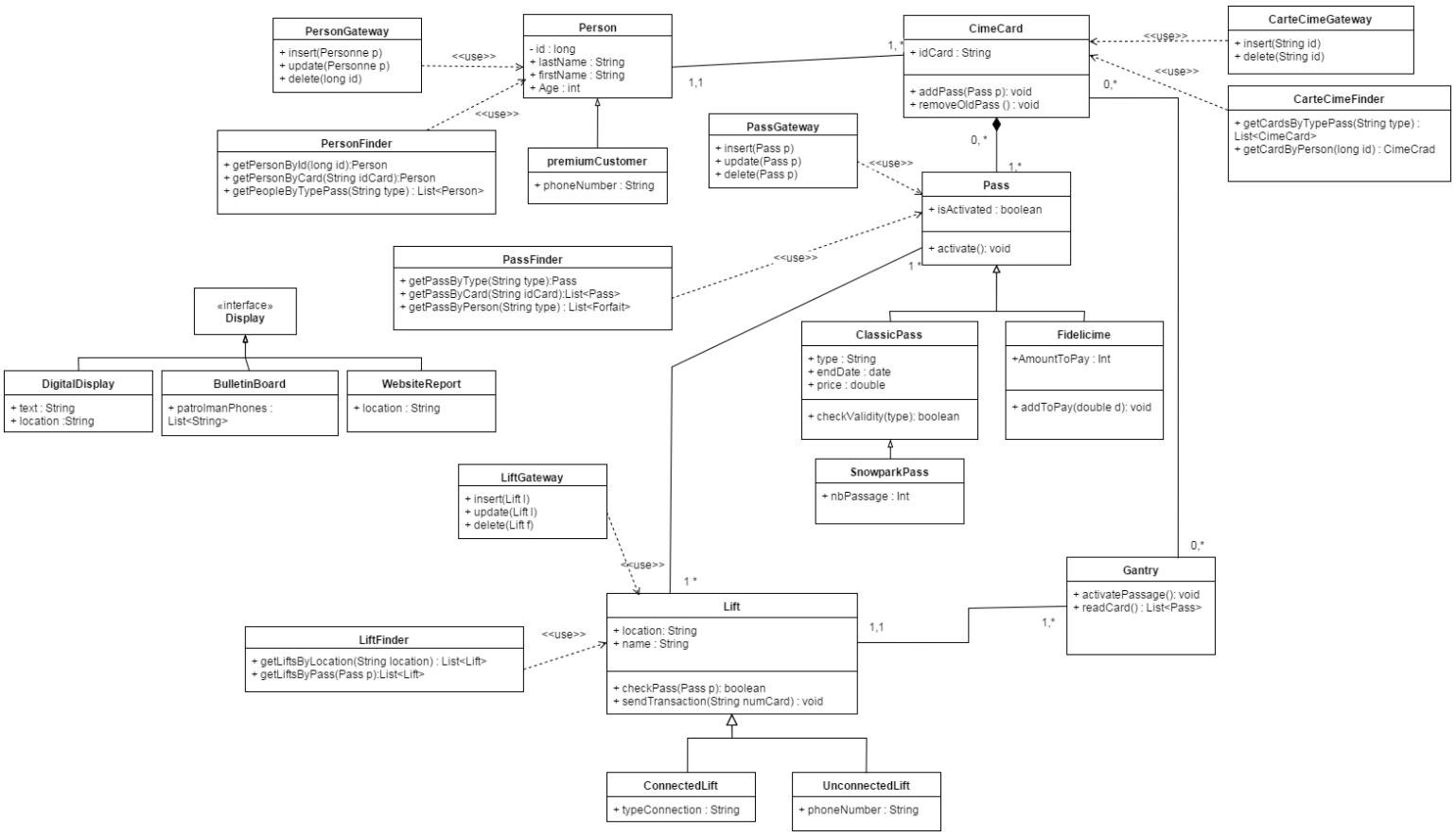

Figure 12: Diagramme de classes après modifications

Vue déploiement

Diagramme de déploiement

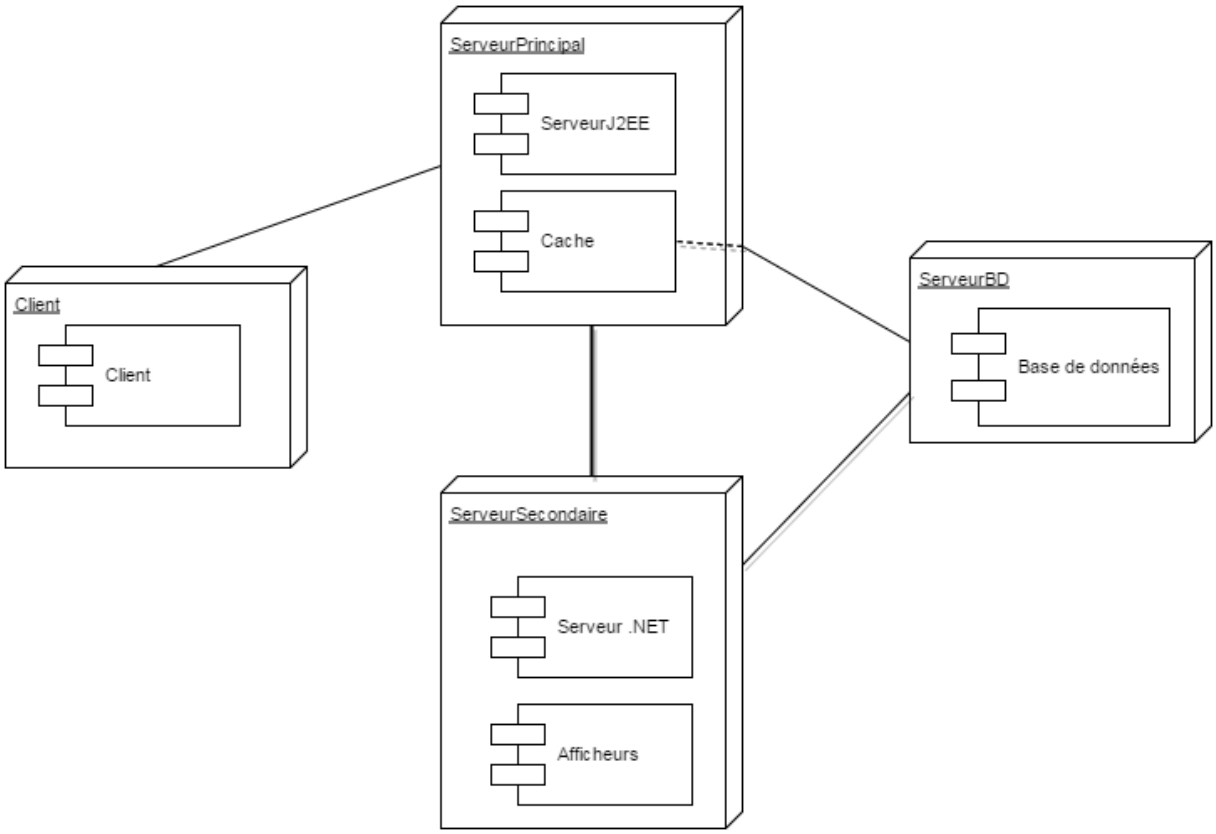

Figure 13: Diagramme de déploiement

Le serveur J2EE est au centre de notre déploiement car c'est lui qui assure les principales fonctionnalités de l'application. Par exemple s'il venait à ne plus être en état de marche, il serait impossible pour les skieurs d'accéder aux remontées mécaniques. C'est pour cela qu'il est seul sur une machine. Celle-ci aura une puissance de calcul supérieure aux autres. Tout en ayant un espace de stockage réservé pour le cache afin de stocker des informations de la base de données. Ainsi si lors d'une après-midi en pleine saison la base de données devenait inaccessible pendant un cours laps de temps, le serveur pourrait toujours faire des lectures dans son cache.
La machine contenant le serveur .NET assurant des services annexes n'utilisera pas toutes les capacités de sa machine. C'est pour cela que nous avons décidé d'ajouter l'hébergement du site web sur cette machine.
La connexion entre les deux serveurs doit être relativement importante afin que le serveur J2EE réponde rapidement aux requêtes des clients.
Enfin, la base de données serait hébergée sur une machine afin d'avoir l'espace de stockage nécessaire et d'être accessible par les services qui en ont besoin (serveur .NET ou les panneaux d'affichage).

Table des illustrations