LIVRABLE 2

Projet « POO »



**MEMBRES DU GROUPE :**

ELBERKENNOU Adel (CG)

BABAALI Amine

BELAIDI Omar

*Sommaire*

* Introduction
* Modélisation
* Langage UML
* Définition des Diagrammes UML
* Modélisation du Besoin
* Conclusion

***INTRODUCTION***

Ce projet a pour objectif de nous faire réaliser une architecture de type client-serveur composée d’une application et d’une base de données. Le domaine auquel appartient cette réalisation est l’informatique de gestion. Cette phase du projet est consacrée à la modélisation logicielle.

***Modélisation logicielle***

* ***Langage UML :***
* Définition :

Le langage UML (Unified Modeling Language, ou langage de modélisation unifié) a été pensé pour être un langage de modélisation visuelle commun, et riche sémantiquement et syntaxiquement. Il est destiné à l'architecture, la conception et la mise en œuvre de systèmes logiciels complexes par leur structure aussi bien que leur comportement. L'UML a des applications qui vont au-delà du développement logiciel, notamment pour les flux de processus dans l'industrie.

Il ressemble aux plans utilisés dans d'autres domaines et se compose de différents types de diagrammes. Dans l'ensemble, les diagrammes UML décrivent la limite, la structure et le comportement du système et des objets qui s'y trouvent.

L'UML n'est pas un langage de programmation, mais il existe des outils qui peuvent être utilisés pour générer du code en plusieurs langages à partir de diagrammes UML. L'UML a une relation directe avec l'analyse et la conception orientées objet.

* **UML et son rôle dans la modélisation et la conception orientées objet :**

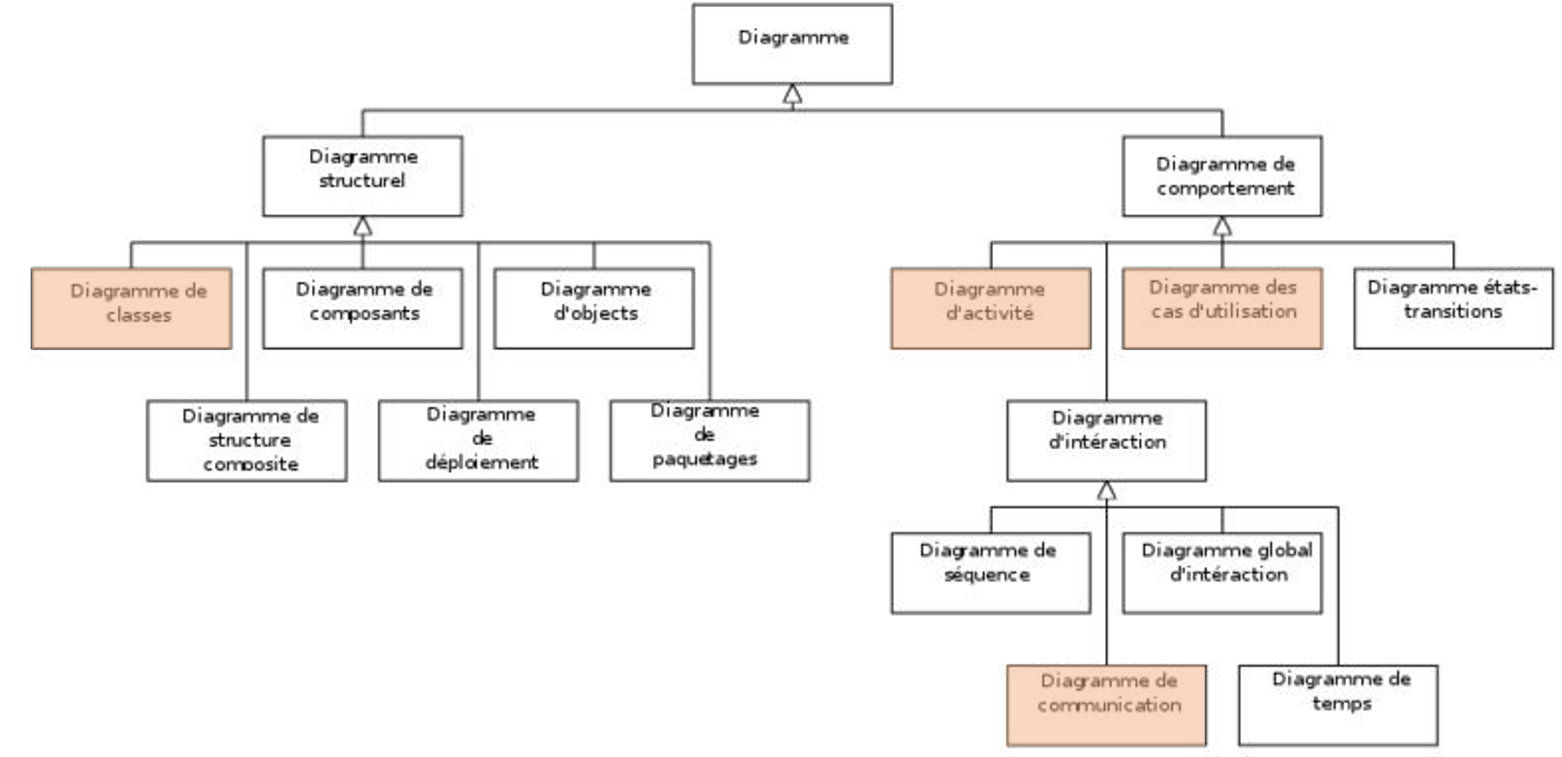
Il existe de nombreux modèles et paradigmes de résolution de problèmes en science informatique, qui est l'étude des algorithmes et des données. On dénombre quatre catégories de modèles de résolution de problèmes : les langages impératifs, fonctionnels, déclaratifs et orientés objet (LOO). Dans le cas des langages orientés objet, les algorithmes sont exprimés en définissant des objets et en les faisant interagir les uns avec les autres. Ces objets sont des éléments que l'on peut manipuler et qui existent dans le monde réel. Il peut s'agir d'immeubles, de widgets sur un ordinateur ou encore d'êtres humains.

Les langages orientés objet dominent le monde de la programmation parce qu'ils modélisent des objets du monde réel. L'UML combine plusieurs notations orientées objet : Object-Oriented Design (conception orientée objet), Object Modeling Technique (technique de modélisation objet) et Object-Oriented Software Engineering (génie logiciel orienté objet).

L'UML utilise les points forts de ces trois approches pour présenter une méthodologie plus cohérente et plus facile à utiliser. Il représente les meilleures pratiques de création et de documentation des différents aspects de la modélisation des systèmes logiciels et d'entreprise.

* ***Définition des Diagrammes UML :***

L'UML utilise des éléments et les associe de différentes manières pour former des diagrammes qui représentent les aspects statiques ou structurels d'un système, ainsi que des diagrammes comportementaux qui capturent les aspects dynamiques d'un système.



**Diagramme de comportement :**

**Diagramme de cas d’utilisation :**

Le diagramme des cas d'utilisation (Use Case Diagram) constitue la première étape de l’analyse UML en :

Modélisant les besoins des utilisateurs.

Identifiant les grandes fonctionnalités et les limites du système.

Représentant les interactions entre le système et ses utilisateurs.

Le diagramme des cas d’utilisation apporte une vision utilisateur et absolument pas une vision informatique. Il ne nécessite aucune connaissance informatique et l’idéal serait qu’il soit réalisé par le client. Le diagramme des cas d’utilisation n’est pas un inventaire exhaustif de toutes les fonctions du système. Il ne liste que des fonctions générales essentielles et principales sans rentrer dans les détails.

Les acteurs :

Avant de rechercher les besoins, la première tâche consiste à définir les limites du système (c.à.d. ce qui est inclus ou pas dans le système), puis à identifier les différentes entités intervenantes sur le système. Ces entités sont appelées acteurs.

Les acteurs se représentent sous la forme d’un petit personnage (stick man) ou sous la forme d’une case rectangulaire (appelé classeur) avec le mot clé « actor ».

Normalement, Il ne peut y avoir qu’un seul acteur principal par cas d’utilisation. En général, l’acteur principal sollicite le cas d’utilisation alors que l’acteur secondaire est sollicité par le cas d’utilisation.

Un acteur est un utilisateur externe au système. Cela peut être :

* Une personne.
* Du matériel (capteurs, moteurs, relais…).
* Un autre système.

Quelquefois, nous utilisons :

* Le stick man si l’acteur est humain
* Le classeur si l’acteur est du matériel ou un autre système

Les liaisons :

Les liaisons permettent de relier les acteurs et les cas d'utilisation. Chaque type de liaison possède une caractéristique et une représentation spécifique à elle.

Nous avons :

Association : Chaque acteur est associé un ou plusieurs cas d’utilisations, la relation d’association peut aussi être appelée relation de communication.

Elle est représentée par un trait reliant l’acteur et le cas d’utilisation.

Nous pouvons rajouter sur ce trait un stéréotype qui va préciser la relation de communication (« communicate »).

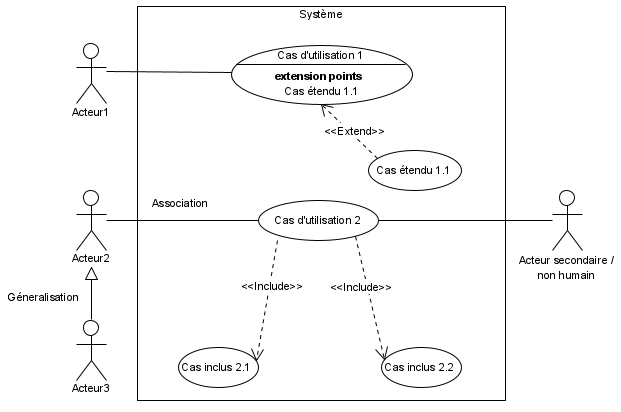
Inclusion (« include ») : La relation d’inclusion sert à enrichir un cas d’utilisation par un autre cas d’utilisation (c’est une sous fonction). La relation est impérative et donc systématique. Dans un diagramme des cas d’utilisation, cette relation est représentée par une flèche pointillée reliant les 2 cas d’utilisation et munie du stéréotype « include».

L’inclusion permet de :

* Partager une fonctionnalité commune entre plusieurs cas d’utilisation.
* Décomposer un cas d’utilisation complexe en décrivant ses sous fonctions.

Extension (« extend ») : Comme la relation d’inclusion, la relation d’extension enrichit un cas d’utilisation par un autre cas d’utilisation de sous fonction mais celui-ci est optionnel. Cette relation est représentée par une flèche en pointillée reliant les 2 cas d’utilisation et munie du stéréotype « extend ». L’extension peut intervenir à un point précis du cas étendu. Ce point s’appelle le point d’extension. Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique point d’extension, et est éventuellement associé à une contrainte indiquant le moment où l’extension intervient. Une extension est souvent soumise à condition.

Généralisation : Lorsque plusieurs acteurs ont besoin d'accéder au même cas d'utilisation, on peut généraliser ces acteurs de façon à les regrouper. De ce fait, le diagramme ne va pas se retrouver bourré de liaisons, et sera plus lisible.



**Diagramme d’activité :**

Diagramme d’activité est utilisé pour afficher la séquence des activités. Les diagrammes d’activité représentent le flux de travail à partir d’un point de départ au point d’arrivée. Détaillant les nombreux sentiers de décision, qui existent dans la progression des événements contenus dans l’activité. Ils peuvent être utilisés à des situations de détail, où le traitement parallèle peut survenir dans l’exécution de certaines activités. Les diagrammes d’activités sont utiles pour la modélisation d’entreprise où ils sont utilisés pour détailler les processus impliqués dans des activités commerciales.

Les sections suivantes décrivent les éléments qui constituent un diagramme d’activité :

Etat d’activité :

L’état d'activité marque une action faite par un objet. Il est représenté par un rectangle arrondi.

Transition :

Quand un état d'activité est accompli, le traitement passe à un autre état d'activité. Les transitions sont utilisées pour marquer ce passage. Les transitions sont modélisées par des flèches.

Etat initial :

L’état initial marque le point d'entrée la première activité. Il est représenté, comme dans le diagramme d'état, par un cercle plein. Il ne peut y avoir qu'un seul état initial sur un diagramme.

Etat final :

L'état final marque la fin du déroulement des opérations.

Bar de synchronisation :

Souvent, certaines activités peuvent être faites en parallèle. Pour dédoubler le traitement "Fork", ou le reprendre quand des activités multiples ont été accomplies ("join"), des barres de synchronisation sont utilisées. Celles-ci sont modélisées par des rectangles pleins, avec des transitions multiples entrantes ou sortantes.

Nœud de décision (decision node) :

Un nœud de décision est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants. Ces derniers sont généralement accompagnés de conditions de garde pour conditionner le choix. Graphiquement, nous représentons un nœud de décision par un losange.

**Diagramme d’interaction :**

**Diagramme de séquence :**

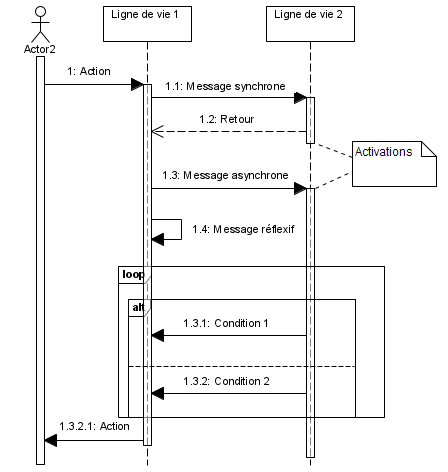
Le diagramme de séquence montre la chronologie des événements et des messages passés entre éléments (lignes de vie) au sein d’une interaction.

La progression temporelle est verticale et les éléments sont représentés horizontalement.

Les éléments sont les entités qui sont en mesure de délivrer des messages. Un utilisateur peut donc figurer dans le diagramme de séquence.

Il est recommandé de garder une certaine cohérence entre le diagramme de cas d'utilisation et le diagramme de séquence. Ainsi, l'acteur sera placé à gauche du diagramme.

Un diagramme simple peut se limiter aux lignes de vie et aux messages, mais souvent il faudra faire appel aux fragments combinés qui permettent d'affiner la représentation.



**Lignes de vie :**

* Comme cela a déjà été annoncé, la ligne de vie décrit une entité du système.
* La progression chronologique se fait du haut vers le bas.
* Lorsque l'entité s'active, le trait en pointillé est remplacé par un trait large.
* Lorsqu'un système n'est plus en fonction toutes les entités sont désactivées.

**Messages :**

Ce sont des éléments de communication unidirectionnelle entre les lignes de vie qui déclenchent une activité dans le destinataire. La réception d’un message provoque un événement chez le récepteur.

La flèche pointillée représente un retour. Cela signifie que le message en question est le résultat direct du message précédent.

Un message synchrone (**émetteur bloqué en attente de réponse**) est représenté par une flèche pleine, alors qu’un message asynchrone est représenté par une flèche évidée.

La flèche qui boucle **(message réflexif)** permet de représenter un comportement interne.

Il s'agit d'une notation qui permet de décrire avec plus de finesse le cycle d'un système :

**Fragments combines :**

Les principaux opérateurs sont :

* loop : boucle. Le fragment peut s’exécuter plusieurs fois, et la condition de garde explicite l’itération ;
* opt : optionnel. Le fragment ne s’exécute que si la condition fournie est vraie ;
* alt : fragments alternatifs. Seul le fragment possédant la condition vraie s’exécutera.

**Diagramme structurel :**

**Diagramme de classe :**

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation.

Alors que le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne.

Il s'agit d'une vue statique, car on ne tient pas compte du facteur temporel dans le comportement du système. Le diagramme de classes modélise les concepts du domaine d'application ainsi que les concepts internes créés de toutes pièces dans le cadre de l'implémentation d'une application. Chaque langage de Programmation orienté objet donne un moyen spécifique d'implémenter le paradigme objet (pointeurs ou pas, héritage multiple ou pas, etc.), mais le diagramme de classes permet de modéliser les classes du système et leurs relations indépendamment d'un langage de programmation particulier.

Les principaux éléments de cette vue statique sont les classes et leurs relations : association, généralisation et plusieurs types de dépendances, telles que la réalisation et l'utilisation.

**Définition de classe :**

Une description d'un groupe d'objets ayant tous des rôles similaires dans le système, qui comprend :

Les caractéristiques structurelles (attributs) définissent quels objets de la classe « savent »

* Représenter l'état d'un objet de la classe
* Sont des descriptions des caractéristiques structurelles ou statiques d'une classe

Les caractéristiques comportementales (opérations) définissent ce que les objets de la classe « peuvent faire »

* Définir la manière dont les objets peuvent interagir
* Les opérations sont des descriptions de caractéristiques comportementales ou dynamiques d'une classe

Notation de classe

Une notation de classe se compose de trois parties :

Nom du cours :

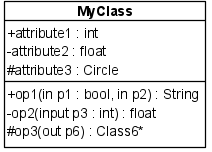
* Le nom de la classe apparaît dans la première partition.
* Attributs de classe :

Les attributs sont affichés dans la deuxième partition.

* Le type d'attribut est affiché après les deux points.
* Les attributs sont mappés sur les variables membres (membres de données) dans le code.

Opérations de classe (méthodes) :

* Les opérations sont affichées dans la troisième partition. Ce sont des services fournis par la classe.
* Le type de retour d'une méthode est indiqué après les deux points à la fin de la signature de la méthode.
* Le type de retour des paramètres de méthode est indiqué après les deux points suivant le nom du paramètre.
* Mappage des opérations sur les méthodes de classe dans le code

La représentation graphique de la classe - MyClass comme indiqué ci-coté :

MyClass a 3 attributs et 3 opérations

Le paramètre p3 de op2 est de type int

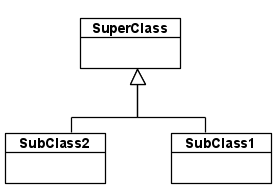
op2 renvoie un float

op3 renvoie un pointeur (désigné par un \*) vers Class6

**Relations de classe**

Une classe peut être impliquée dans une ou plusieurs relations avec d'autres classes. Une relation peut être de l'un des types suivants : (Reportez-vous à la figure de droite pour la représentation graphique des relations).

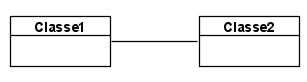
**Type de relation :**

Héritage (ou généralisation) :

Représente une relation « est-un ».

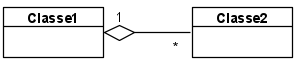
* Un nom de classe abstrait est affiché en italique.
* SubClass1 et SubClass2 sont des spécialisations de Super Class.
* Une ligne continue avec une pointe de flèche creuse qui pointe de l'enfant vers la classe parent

Association simple :

* Un lien structurel entre deux classes homologues.
* Il existe une association entre Classe1 et Classe2
* Une ligne continue reliant deux classes

Agrégation :

Un type particulier d'association. Cela représente une « partie de » relation.

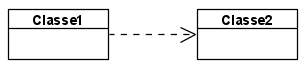
* Class2 fait partie de Class1.
* De nombreuses instances (désignées par \*) de Class2 peuvent être associées à Class1.
* Les objets de Classe1 et Classe2 ont des durées de vie distinctes.
* Une ligne continue avec un diamant non rempli à l'extrémité de l'association connecté à la classe de composite

Composition :

Un type spécial d'agrégation où les pièces sont détruites lorsque le tout est détruit.

* Les objets de la classe 2 vivent et meurent avec la classe 1.
* Class2 ne peut pas être autonome.
* Une ligne continue avec un diamant rempli à l'association connectée à la classe de composite

Dépendance :

* Existe entre deux classes si les modifications apportées à la définition de l'une peuvent entraîner des modifications de l'autre (mais pas l'inverse).
* La classe 1 dépend de la classe 2
* Une ligne pointillée avec une flèche ouverte

**Visibilité des attributs de classe et des opérations :**

Dans la conception orientée objet, il existe une notation de la visibilité pour les attributs et les opérations. UML identifie quatre types de visibilité : publique, protégée, privée et package.

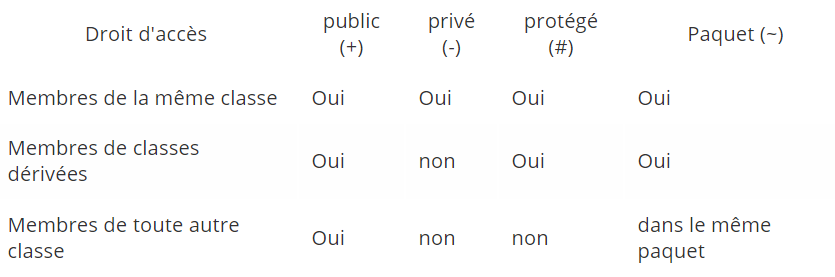
Les symboles +, -, # et ~ avant un attribut et un nom d'opération dans une classe indiquent la visibilité de l'attribut et de l'opération.

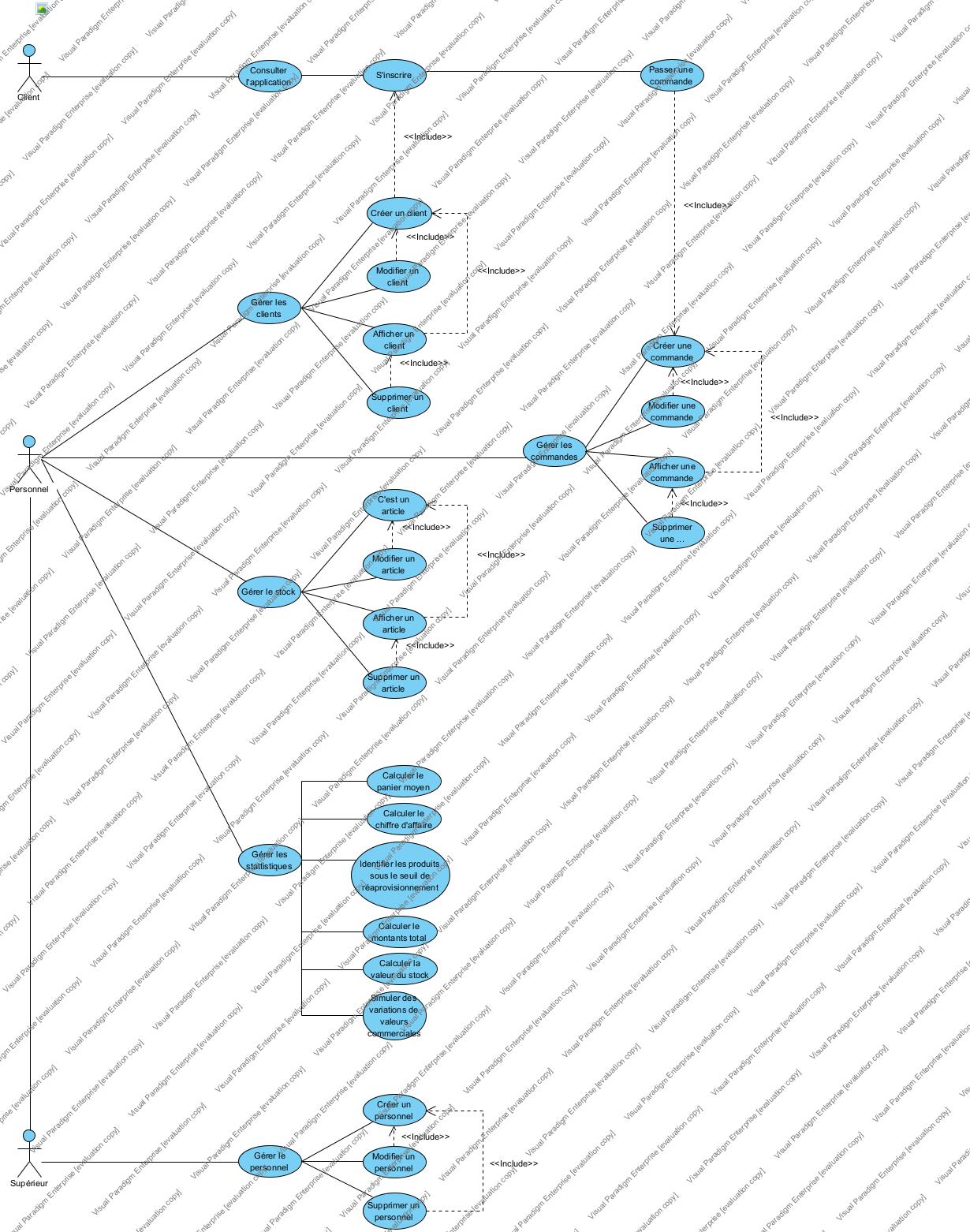
+ désigne des attributs ou des opérations publics

- désigne des attributs ou des opérations privés

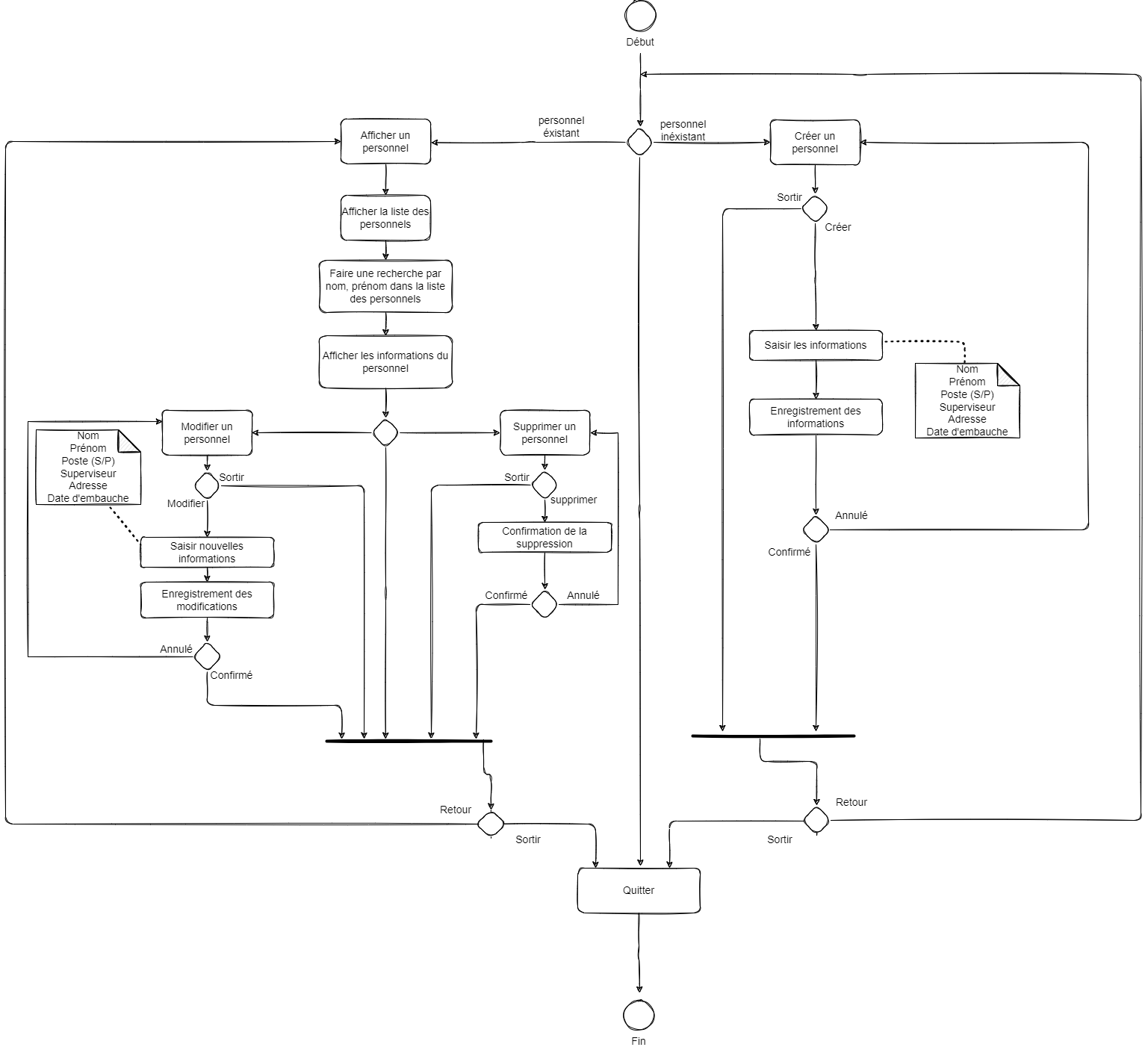
# désigne des attributs ou des opérations protégés

~ désigne les attributs ou les opérations du package



* ***Modélisation Logicielle du Besoin :***
* ***Diagramme Use Case :***

Dans notre cas, nous avons trois acteurs :

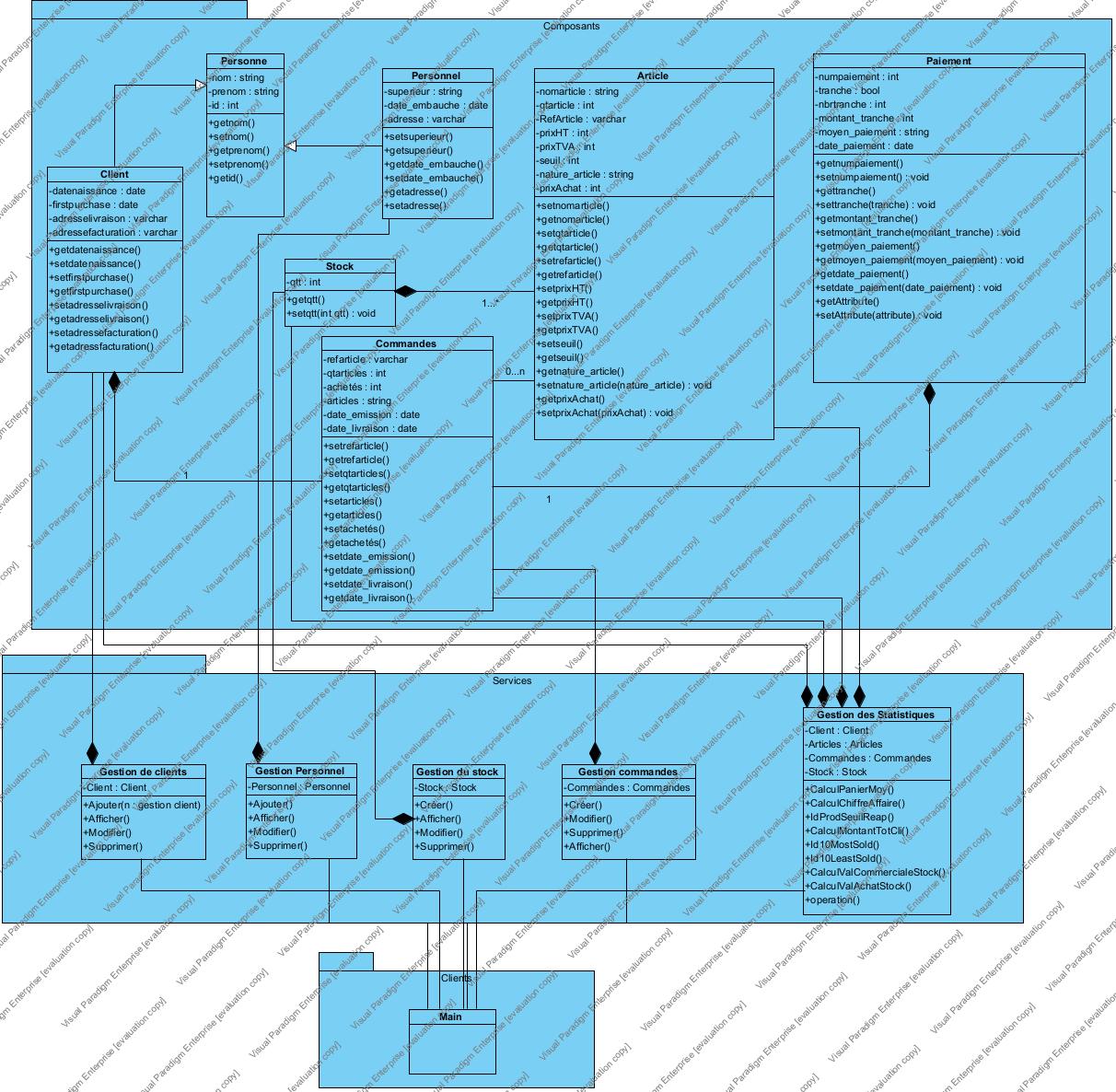
* Le client : Il a la possibilité de consulter le catalogue, de s’inscrire au site et de passer une commande
* Le personnel : Il peut gérer (créer, modifier, supprimer et afficher) les comptes clients, le stock, les commandes des clients et la gestion des statistiques (calcule panier moyen, calcule chiffre d’affaires...)
* Le supérieur : Il gère le personnel, cela veut dire qu’il crée, modifie et supprime les personnels
* ***Diagramme D’Activité***

Notre diagramme d'activité est utilisé pour documenter le déroulement des opérations dans notre système, l’interface gestion personnel contient différentes actions qui vont servir à :

* Les affectations : C'est-à-dire, le responsable personnel pourra connaître les informations concernant les employées dans tous les services avec l’action ‘afficher personnel’.
* Les inscriptions : permet de contrôler les personnels qui peuvent être inscrit dans le système de gestion du personnel de la base de données avec l’action ‘créer un personnel’ qui va être suivre par des actions pour saisir les informations pour compléter la création et l’enregistrer ou bien l’annuler.
* Les modifications : sert pour la modification. C'est-à-dire donné la possibilité de changer les différentes informations concernant le personnel.
* Les suppressions : sert pour la suppression. Donc mettre à jour l’information dans la base de données.

Note :

Poste(S/P) :pour définir si la personne est Supérieur (S) ou Personnel (P)

* ***Diagramme de Classe :***

Notre diagramme de classe se compose de classes reliées les unes aux autres avec différentes relations.

Nous avons tout d’abord créé une classe « Personne » qui contient dans sa partie attributs : un nom, un prénom et un ID, déclarée avec le symbole # pour définir leurs visibilité sur « Protected ».

Pour chacun de ses attributs, nous avons déclaré dans la partie opérations des accesseurs (getter/setter) qui vont servir à modifier/récupérer la valeur d’un attribut, un constructeur (qui porte le même nom de la classe et qui servira par la suite à l’initialisation des données membres)

On a utilisé dans notre diagramme de classe :

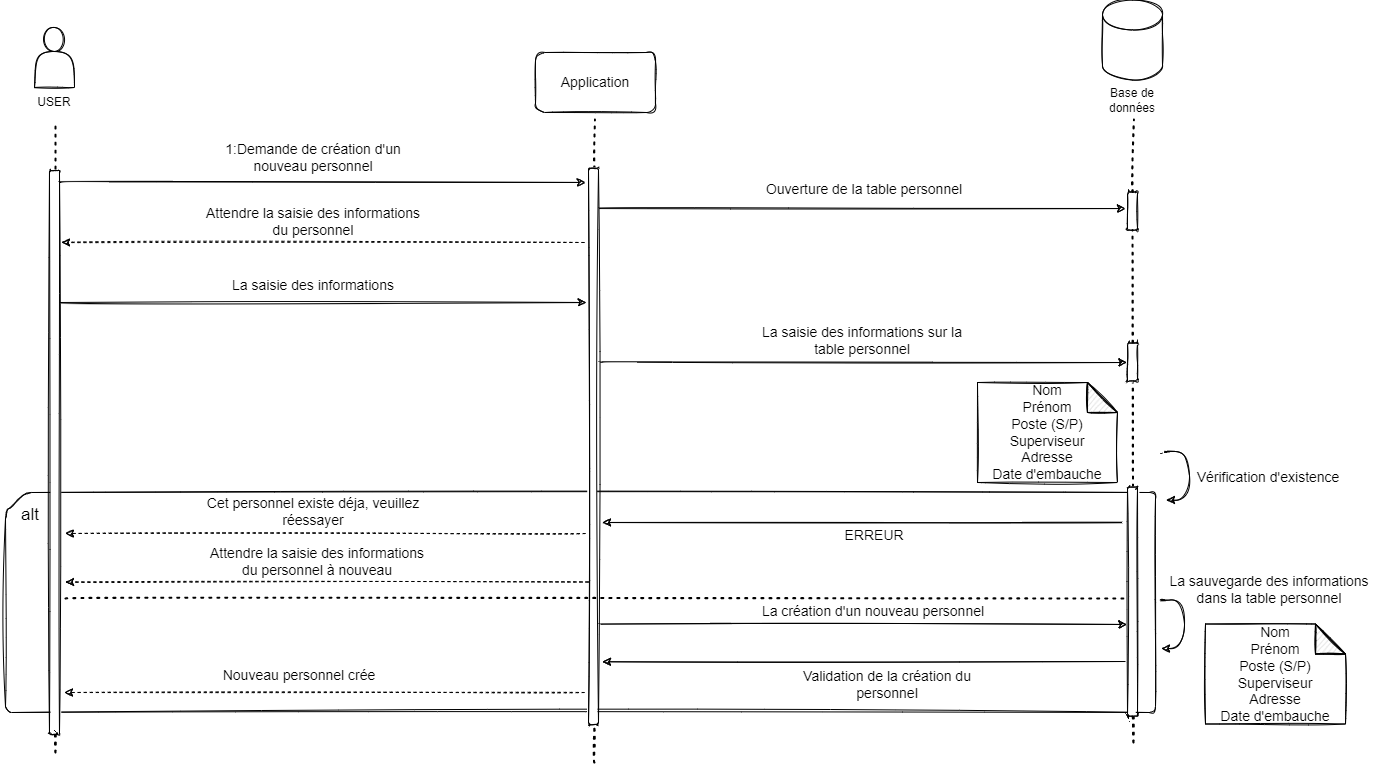
* Des Agrégations (de type composition) : comme entre la classe « gestion de clients » et la classe « Client »
* Héritage (Généralisation) : où les classes « Personnel » et « Client » sont des spécialisations de la classe « Personne »
* ***Diagrammes de Séquence :***

Nous avons mis en place quatre diagrammes de séquence pour le cas d’utilisation « Gestion du personnel » qui montrent la chronologie des événements et des messages passés entre l’Admin et le système au sein d’une interaction :

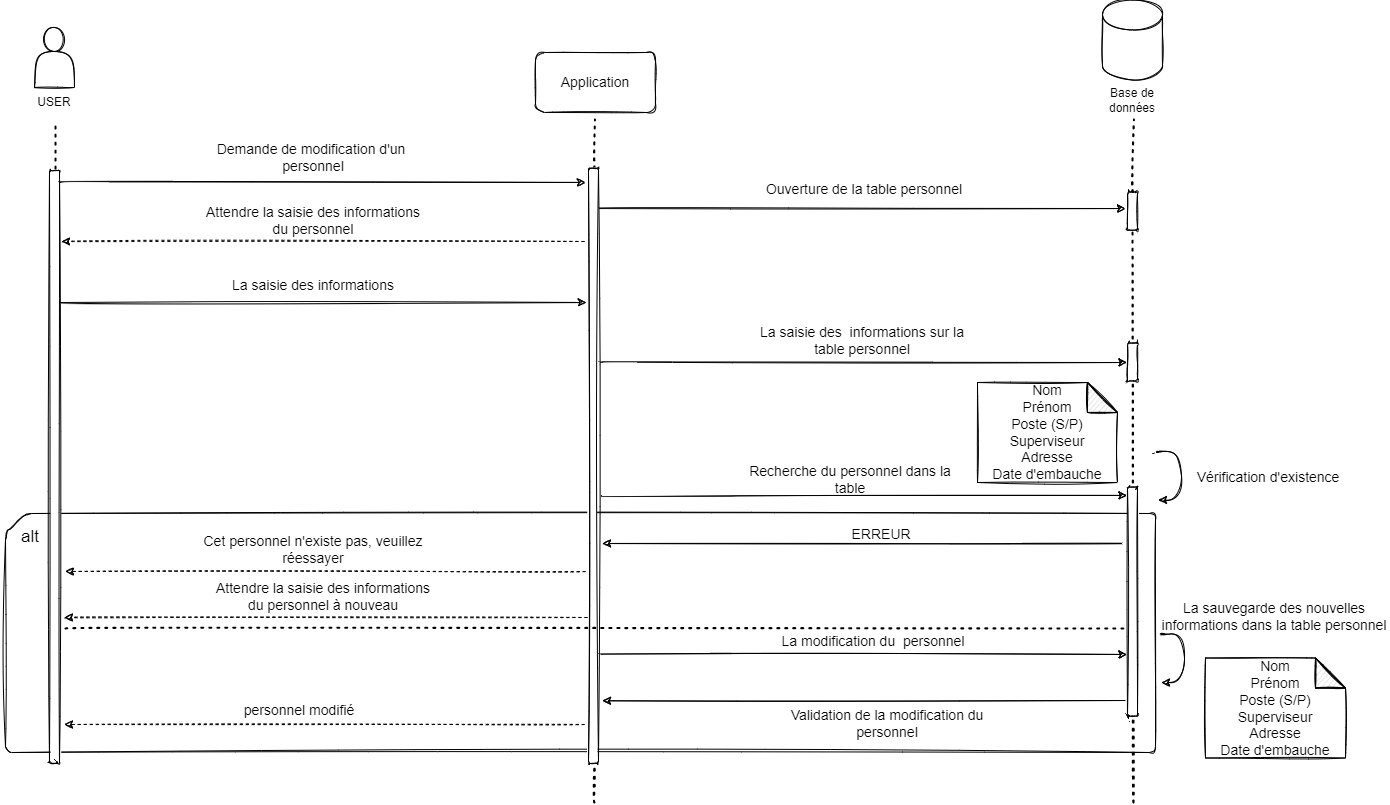
1. Création d’un personnel
2. Modifier un personnel
3. Afficher un personnel
4. Supprimer un personnel

Ces derniers sont montrés ci-dessous :

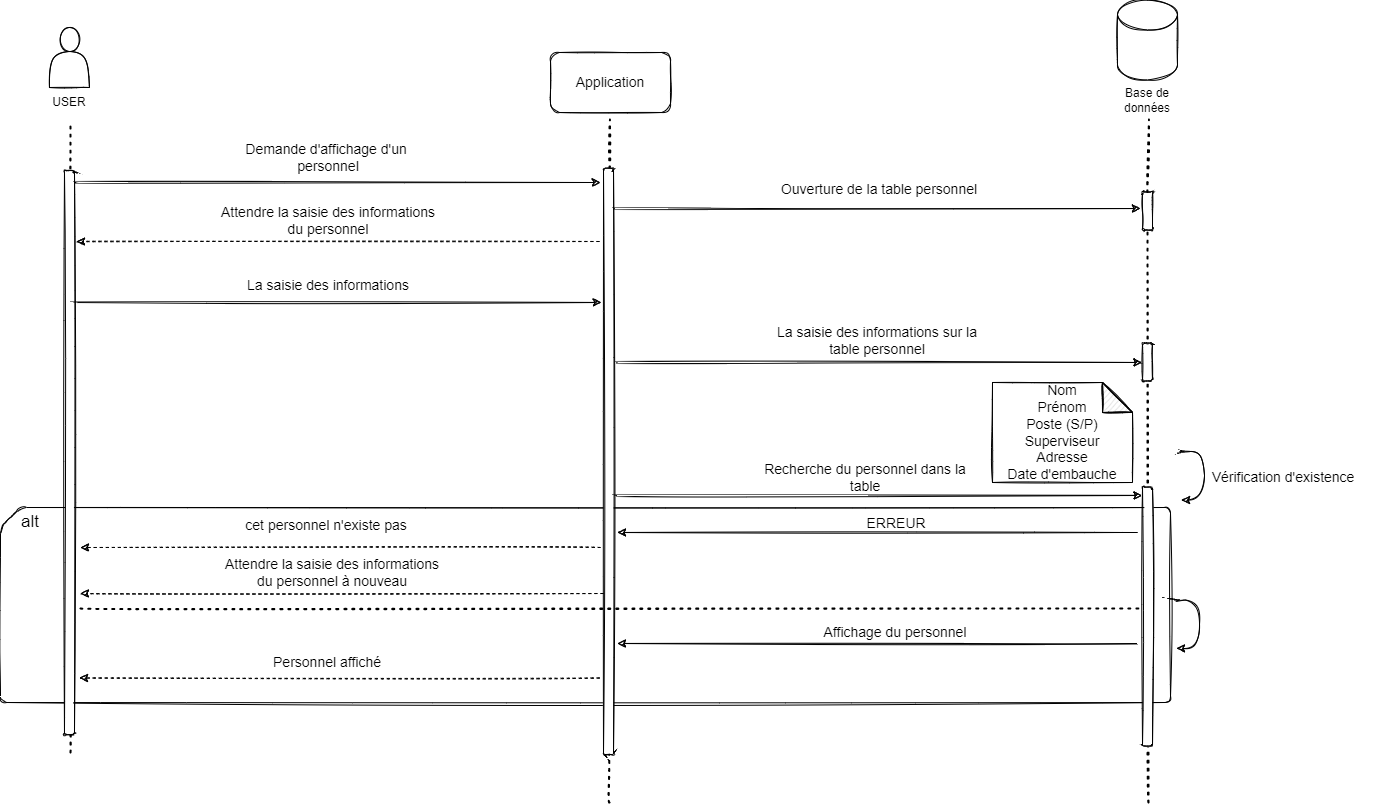
* Création d’un personnel :



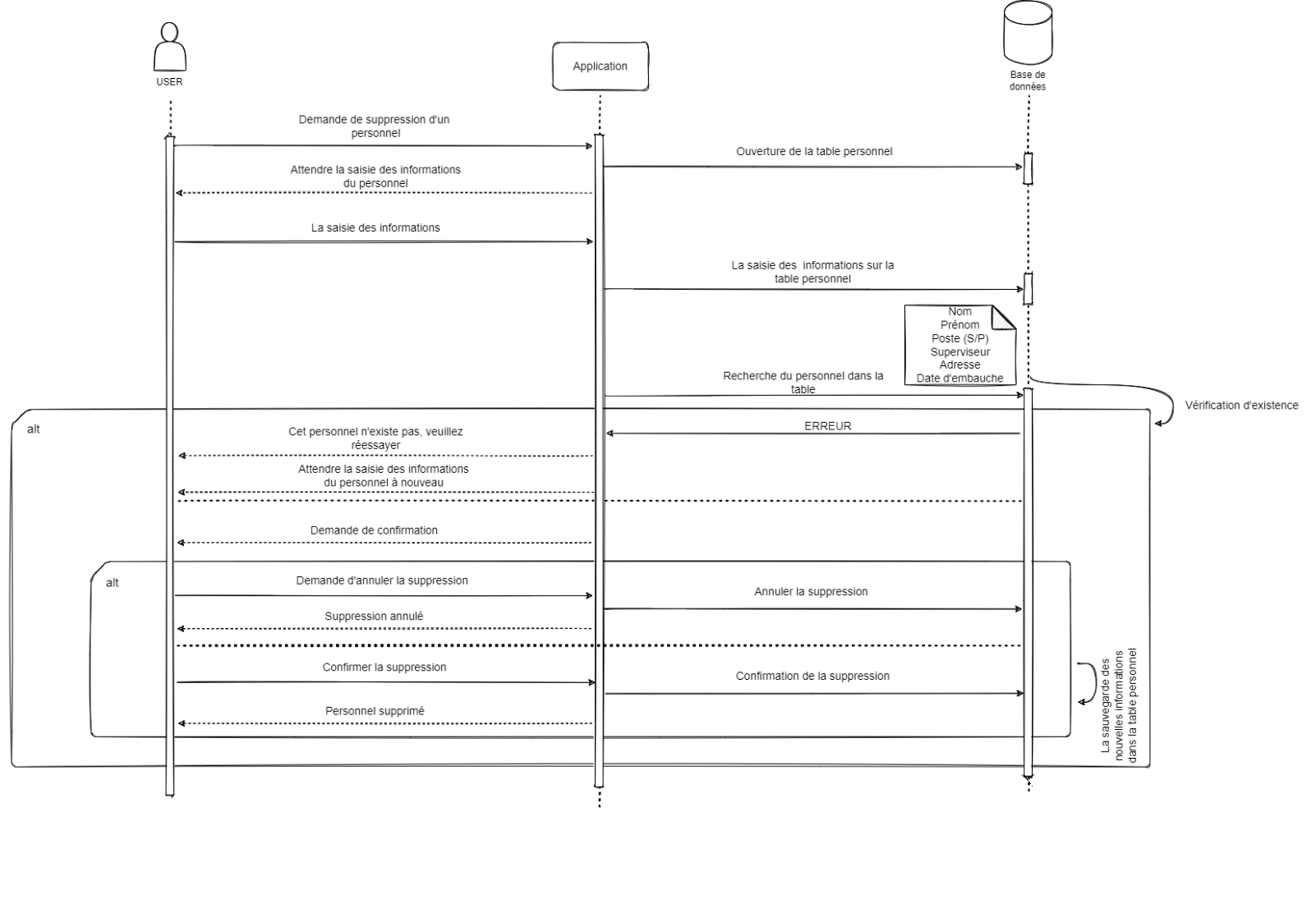
* Modification d’un personnel :



* Affichage d’un personnel :



* Suppression d’un personnel :



***Conclusion***

**En conclusion, nous avons réussi à réaliser la modélisation et à décrire de manière graphique les besoins et les solutions fonctionnelles et techniques de notre projet logiciel.**