**DEUXIEME PARTIE : ETUDE DETAILLEE**

**Chapitre 3 : CADRE METHDOLOGIQUE**

Le rôle principal de l’informatique est de faciliter la vie des gens en automatisant les taches. Un projet informatique comme notre mémoire doit analyser les besoins de ses utilisateurs et de les traduire en un langage qui soit compréhensible par la machine pour après être présenté sous forme d’interface utilisable par l’Homme à travers l’interface Homme-Machine. Ainsi pour qu’il y ait une cohérence dans tout cela un langage de modélisation un incontournable.

**Section 1 : THEORIE DE CONCEPTION**

1. **Approche fonctionnelle**

Les méthodes fonctionnelles sont originaires des langages procéduraux (Pascal, C…). Elles mettent en évidence les fonctions à assurer et proposent une approche hiérarchique descendante et modulaire. Ces méthodes utilisent intensivement les raffinements successifs pour produire des spécifications dont l’essentiel est sous forme de notation graphique sous forme de diagrammes de flots de données. Le plus haut niveau représente l’ensemble du problème. Chaque niveau est ensuite décomposé en respectant les entrées et sorties du niveau supérieur. La décomposition se poursuit jusqu’à arriver à des composants maitrisables.

1. **Avantages de la modélisation fonctionnelle**

Cette approche a un avantage certain appelé la factorisation des comportements (C’est-à-dire que pour créer une fonction d’une application, rien ne vous empêche d’utiliser un autre ensemble de fonctions (Réutilisation de code)). Il faut aussi signaler qu’avec cette approche l’exécution en devient plus rapide car le code est léger par rapport aux autres approches.

1. **Inconvénients de la modélisation fonctionnelle**

Parmi les inconvénients de la modélisation fonctionnelle on peut citer :

* Un problème de sécurité causé par une séparation des données et traitements.
* Une maintenance complexe en cas d’évolution d’une application (Une simple mise à jour de l’application à un point donné peut impacter en cascade sur d’autres fonctions de l’application). On devra donc mettre toute l’application à jour.
* Il faut aussi signaler que le codage (programmation) est fastidieux car avec cette approche il n’y a pas de séparation de code donc cela devient brouillon.
* L’absence d’environnement intégré quant à la génération automatique de code et a une aide poussée pour la correction des erreurs de code.

1. **Approche objet**

L’approche orienté objet fait une séparation de l’application comme une collections de classe qu’on peut considérer comme des moules a objets dissociées, identifiées et possédant des propriétés différentes. Une propriété est soit un attribut (i.e. une donnée caractérisant l’état de l’objet), soit une entité comportementale de l’objet (i.e. une fonction). La fonctionnalité du logiciel émerge alors de l’interaction entre les différents objets qui le constituent. L’une des particularités de cette approche est qu’elle rapproche les données et leurs traitements associés au sein d’un unique objet.

Elle se base essentiellement sur trois concepts que sont :

* L’encapsulation
* L’héritage
* Le polymorphisme

**Encapsulation** : regroupement d’attributs et de méthodes au sein d'une même entité appelée « classe ».

**Héritage** : permet de définir une hiérarchie de classe, chaque classe fille héritant des méthodes et des données de son/ses antécédent(s).

**Polymorphisme** : deux objets héritant d’une même méthode d'une classe parente, peuvent réagir de façon différente à l'appel de cette méthode et, à cette fin, redéfinir la méthode. Il est ensuite possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque.

1. **Avantages de la modélisation orientée objet**
   * Programmes plus faciles à maintenir : Si on décide de modifier la structure des données dans un programme séquentiel, presque tout le code est à réécrire
   * Programmation plus claire : Les fonctions sont rattachées à un type de données
   * Modularité accrue : Possibilité de réutiliser le code.
   * Facilité de maintenance : Pour modifier une fonctionnalité on modifie juste la classe de base pas besoin de parcourir tout le code.
   * Facilité de mise à jour : On ajoute facilement une autre fonctionnalité en ajoutant une nouvelle méthode.
2. **Inconvénients de la modélisation orientée objet**
   * Le programme résultant peut être moins efficace (en termes de taille mémoire et de rapidité).
3. **Choix d’une méthodologie d’analyse et de conception**

Selon nos besoins et les spécificités de notre projet on pourrait être amené à choisir l’approche fonctionnelle ou l’approche objet. Comme nous cherchons la précision et que notre application utilisera d’autre fonctionnalités qui sont prêtes pour l’utilisation on va plutôt utiliser l’approche objet plus précisément UML qui est un langage de modélisation parmi tant d’autres.

Outre l’avantage, la possibilité de pouvoir gérer chaque phase tant bien dans le temps qu’au côté ressource-lui confère un réel avantage. C’est un des meilleurs outils de modélisation qui est adapté à notre étude.

**Section 2 : PRESENTATION ET MODELISATION EN UML**

UML, c’est l’acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un langage visuel constitué d’un ensemble de schémas, appelés des diagrammes, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour représenter le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d’être effectuées par le logiciel, etc.

Réaliser ces diagrammes revient donc à modéliser les besoins du logiciel à développer.

Et ces diagrammes, on les réalise comment ?

On a le choix ! On peut soit reprendre les normes de diagrammes et les dessiner à la main, soit utiliser des logiciels gratuits ou payants pour les réaliser. On peut utiliser Astah Community, ArgoUml, BoUml, PowerDesigner, StarUml etc.

Petite historique :

UML est né de la fusion des trois méthodes qui ont influencé la modélisation objet au milieu des années 90 : OMT, Booch et OOSE. Il s’agit d’un compromis qui a été trouvé par une équipe d'experts : Grady Booch, James Rumbaugh et Ivar Jacobson. UML est à présent un standard défini par l'Object Management Group (OMG). De très nombreuses entreprises de renom ont adopté UML et participent encore aujourd’hui à son développement.

UML est surtout utilisé lorsqu’on prévoit de développer des applications avec une démarche objet (développement en Java, en C++, en PHP, etc.).

On peut tout à fait s’en servir pour décrire de futures applications, sans pour autant déjà être fixé sur le type de développement.

Le langage UML ne préconise aucune démarche, ce n’est donc pas une méthode. Chacun est libre d’utiliser les types de diagramme qu’il souhaite, dans l’ordre qu’il veut. Il suffit que les diagrammes réalisés soient cohérents entre eux, avant de passer à la réalisation du logiciel.

En modélisation UML on utilise plusieurs types de diagrammes parmi lesquels on peut citer :

* **Diagramme de cas d’utilisation**

Le diagramme de cas d’utilisation représente les fonctionnalités (ou cas d’utilisation) nécessaires aux utilisateurs. On peut faire un diagramme de cas d’utilisation pour le logiciel entier ou pour chaque package.

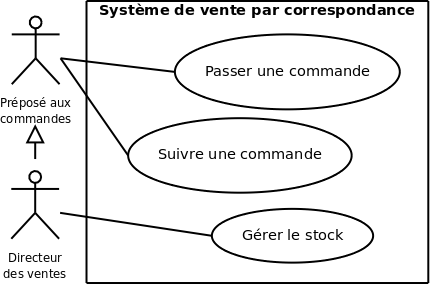


Figure 1 : Exemple d’un diagramme de cas d’utilisation

Détaillons les composants des diagrammes de cas d’utilisation.

**Acteur :** Un acteur est une entité extérieure au système modélisé et qui interagit directement avec lui.

**Cas d’utilisation :** Un cas d'utilisation (en anglais use case) permet de mettre en évidence les relations fonctionnelles entre les acteurs et le système étudié.

**Relation d’association :** Une association représente une relation sémantique entre les objets d'une classe.

**Inclusion :** une relation d'inclusion est une relation dans laquelle un cas d'utilisation (le cas d'utilisation de base) inclut les fonctionnalités d'un autre cas d'utilisation (le cas d'utilisation inclus).

**Extension :** Comme la relation d’inclusion c’est aussi quand un cas d’utilisation étend les fonctionnalités d’un autre cas d’utilisation sauf qu’ici le cas d’utilisation de base n’est pas indispensable.

**Généralisation : U**ne relation de généralisation est une relation dans laquelle un élément de modèle (l'enfant) est basé sur un autre élément de modèle (le parent). Les relations de généralisation sont utilisées dans les diagrammes de classes, de composants, de déploiement et de cas d'utilisation pour indiquer que l'enfant reçoit tous les attributs, opérations et relations qui sont définis dans le parent.

* **Diagramme de classe**

Le diagramme de classes

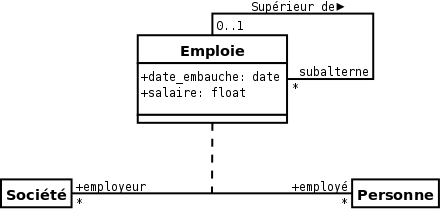
Dans la phase d’analyse, ce diagramme représente les entités (des informations) manipulées par les utilisateurs.   
Dans la phase de conception, il représente la structure objet d’un développement orienté objet.  


Figure 2 : Exemple d’un diagramme de Classe

Détaillons les composants des diagrammes de Classes.

**Classe :** Une classe est un concept abstrait représentant des éléments variés comme : des éléments concrets (ex. : des avions), des éléments abstraits (ex. : des commandes de marchandises ou services).

**Agrégation :** En programmation informatique et plus précisément en programmation orientée objet, l'agrégation permet de définir une entité comme étant liée à plusieurs entités de classe différentes.

**Composition :** Un lien de composition symbolise l'existence d'une [agrégation](https://fr.wikipedia.org/wiki/Agr%C3%A9gation_(programmation)) particulière, dite « forte », entre deux entités (classes).Une composition est définie par les points suivants : Durée de vie : toute classe agrégée est détruite quand la classe composite est détruite. Exclusivité : une classe agrégée ne peut l'être que par une seule classe composite. Notion de « fait partie de ».

**Héritage :** En programmation orientée objet, l'héritage est un mécanisme qui permet, lors de la déclaration d'une nouvelle classe, d'y inclure les caractéristiques d'une autre classe.

**Association :** Une association représente une relation possible entre les objets d'une classe. Une relation de composition décrit une relation de contenance et d'appartenance.

**Cardinalité :** La cardinalité ou multiplicité, dans les schémas relationnels en modélisation des données, sert à compter le nombre minimum et maximum de possibilités que chaque classe contient dans la relation liant 2 ou plusieurs objets.

* **Diagramme de séquences**

Le diagramme de séquence permet de décrire les différents scénarios d’utilisation du système.

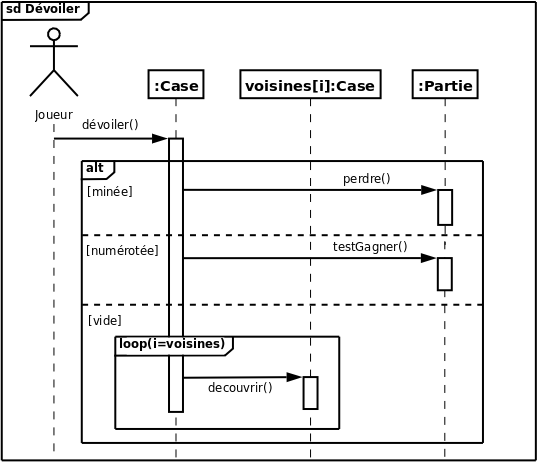


Figure 3 : Exemple d’un diagramme de Séquence

Détaillons les composants des diagrammes de Séquence.

**Messages :** Un message est un élément de diagramme Unified Modeling Language (UML) qui définit un type particulier de communication entre les instances au cours d'une interaction.

**Messages minutés :** un message minuté (timeout) Bloque l'expéditeur pendant un temps donné (qui peut être spécifié dans une [contrainte](http://uml.free.fr/cours/p16.html#cont)), en attendant la prise en compte du message par le récepteur. L'expéditeur est libéré si la prise en compte n'a pas eu lieu pendant le délai spécifié.

**Messages Synchrones :** Bloque l'expéditeur jusqu'à prise en compte du message par le destinataire. Le flot de contrôle passe de l'émetteur au récepteur (l'émetteur devient passif et le récepteur actif) à la prise en compte du message

**Messages Asynchrones :** N'interrompt pas l'exécution de l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré (jamais traité)

**Ligne de vie :** Une ligne de vie représente un objet dans l'interaction.  Elle peut être une représentation d'une instance.

* **Diagramme de composants**

Le diagramme de composants décrit tous les composants utiles à l’exécution du système (applications, librairies, instances de base de données, exécutables, etc.). Ce diagramme ne sera pas étudié dans ce cours.

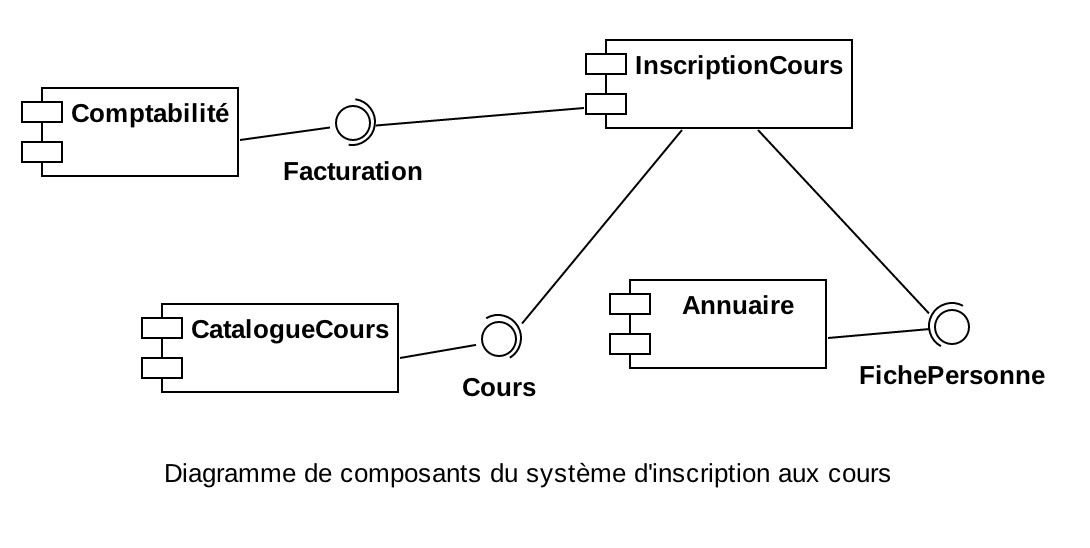


Figure 4 : Exemple d’un diagramme de Composants

* **Diagramme d’objet**

Le diagramme d’objets sert à illustrer les classes complexes en utilisant des exemples d’instances.

Une instance est un exemple concret de contenu d’une classe. En illustrant une partie des classes avec des exemples (grâce à un diagramme d’objets), on arrive à voir un peu plus clairement les liens nécessaires. Ce diagramme ne sera pas étudié dans ce cours.

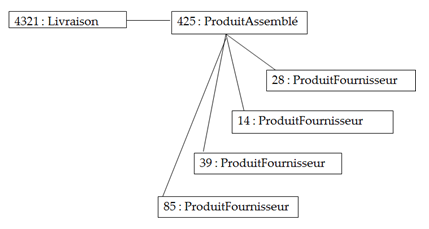


Figure 5 : Exemple d’un diagramme d’Objet

* **Diagramme de déploiement**

Le diagramme de déploiement correspond à la description de l’environnement d’exécution du système (matériel, réseau…) et de la façon dont les composants y sont installés. Ce diagramme ne sera pas étudié dans ce cours.

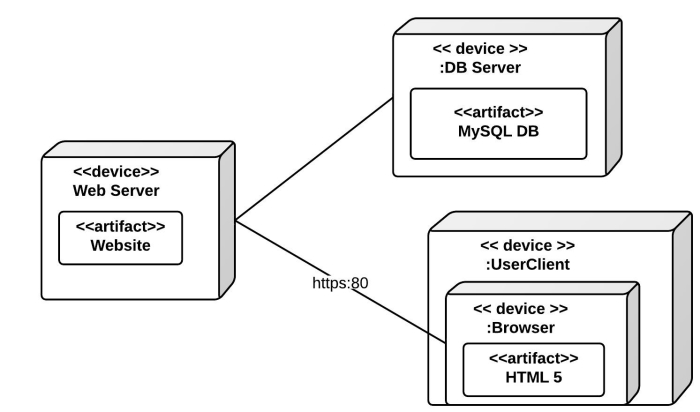


Figure 6 : Exemple d’un diagramme de Déploiement

**Chapitre 4 : ETUDE FONCTIONNELLE**

**Section 1 : IDENTIFICATION DES BESOINS**

Les cas d’utilisation de notre application dépendent des objectifs que l’on s’est fixé ci-dessus. Et qui dit besoin dit cas d’utilisation.

* **Identification des cas d’utilisation**

L’application devra permettre aux utilisateurs qui sont : Les clients qui pourront regarder les produits disponibles, créer un compte, résilier leurs comptes, demander un produit…, les producteurs qui pourront créer un compte, résilier leurs comptes, publier un produit, regarder les produits demandés…, l’administrateur qui pourra supprimer des publications, arranger les affichages, bloquer ou débloquer un compte… Nous allons tout d’abord commencer par identifier les acteurs.

1. **Les acteurs**

Les acteurs qui vont utiliser le système sont :

1. **L’administrateur**

C’est l’utilisateur qui est chargé de toute l’organisation au sein de notre application. Il sera chargé de gérer les utilisateurs, les publications, les conseils. Il pourra supprimer une publication ou un conseil ou une demande.

1. **Le client**

Il pourra se connecter pour acheter un produit ou consulter les produits ou même voir les conseils.

1. **Le producteur**

Il pourra se connecter pour acheter un produit ou consulter les produits ou même voir les conseils. Il pourra aussi consulter les statistiques pour voir quels sont les produits qui sont demandés le plus souvent.

1. **Les cas d’utilisation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Cas d’utilisation** | **Acteur** | **Message Emis** | **Message Reçu** |
| **Publier Produit** | **Producteur** | **Ajouter/Annuler** |  |
| **Modifier publication** | **Producteur/Admin** | **Modifications** |  |
| **Demander Produit** | **Acheteur** | **Demander/Annuler** |  |
| **Acheter Produit** | **Acheteur** | **Acheter/Annuler** |  |
| **Consulter Produits** | **Tous** | **Consulter** | **Liste Produits** |
| **S’inscrire** | **User** |  |  |
| **Se Connecter** | **User** |  |  |
| **Bloquer Compte** | **Admin** | **Bloquer** |  |
| **Modifier User** | **User/Admin** |  |  |
| **Consulter Conseils** | **User/Admin** |  | **Liste conseils** |
| **Ajouter Conseils** | **Admin** | **Conseil** |  |
| **Supprimer Conseils** | **Admin** |  |  |
| **Supprimer Publication** | **Admin/Producteur** |  |  |
| **Rechercher Produit** | **Tous** |  | **Produits** |

**Tableau 1 : Liste de cas d’utilisation**

**Section 2 : DIAGRAMMES**

1. **Diagrammes de cas d’utilisation**
2. **Processus de Publication de Produit**

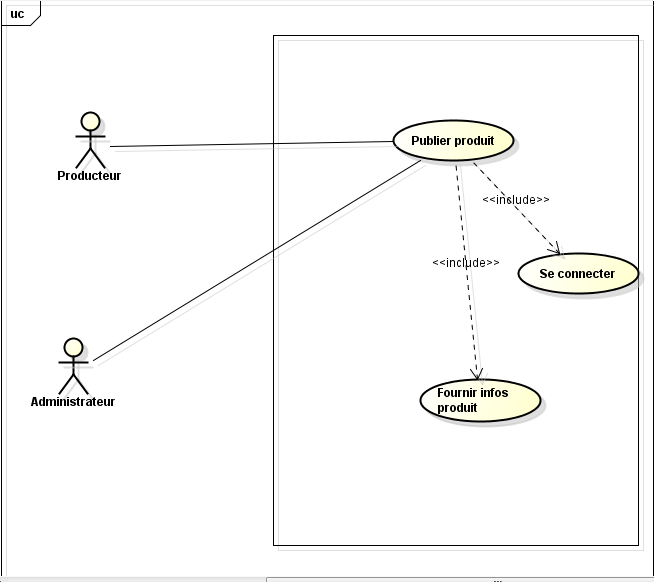
****

Figure 7 : Diagramme de cas d’utilisation du processus de publication de produit.

1. **Processus D’Achat de Produit**

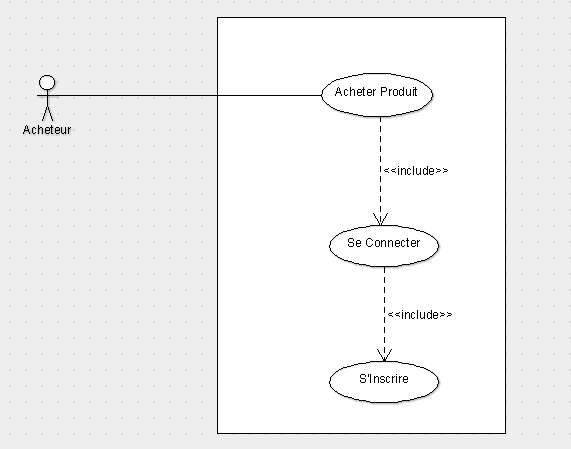
****

Figure 8 : Diagramme de cas d’utilisation du processus d’achat d’un produit

1. **Diagramme complet**

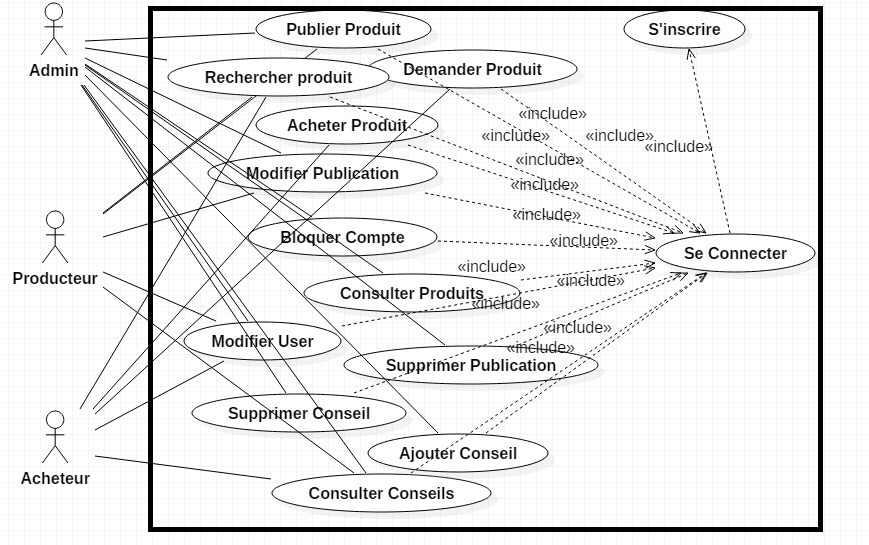
****

Figure 9 : Diagramme de cas d’utilisation de notre application

1. **Diagrammes de Séquences**
2. **Processus de publication de produit**

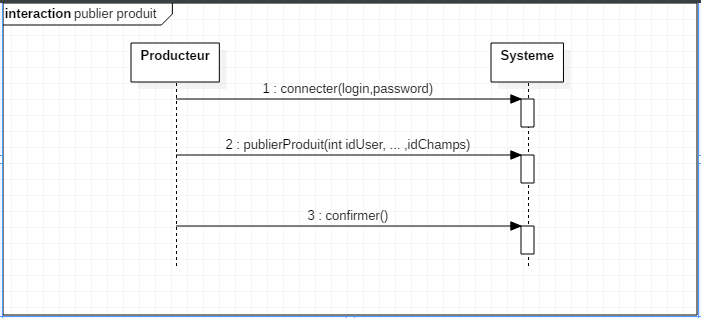
****

Figure 10 : Diagramme de séquences du processus de publication de produit

1. **Processus d’achat de produit**

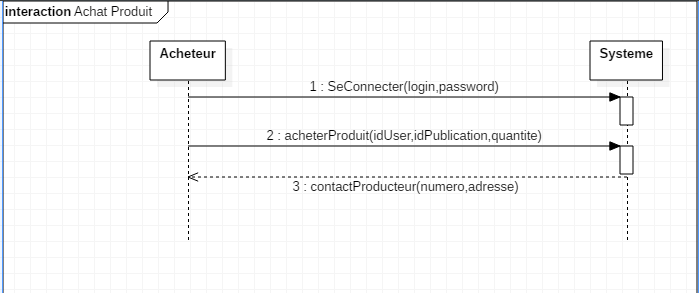
****

Figure 11 : Diagramme de séquences du processus d’achat de produits

1. **Diagrammes d’Objets**
2. **Publication d’un produit**

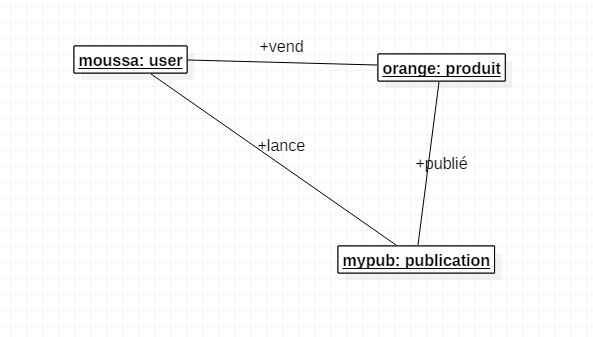
****

Figure 12 : Diagramme d’objets du processus de publication de produit

1. **Diagrammes de déploiement**

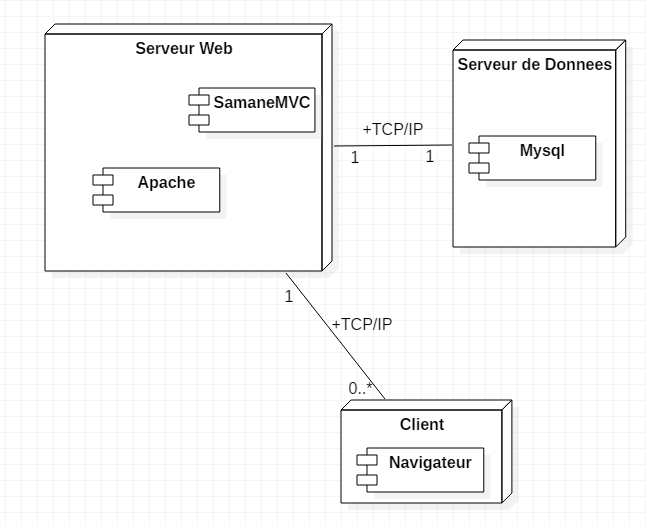
****

Figure 13 : Diagramme de déploiement de notre plateforme

1. **Diagramme de Classes**

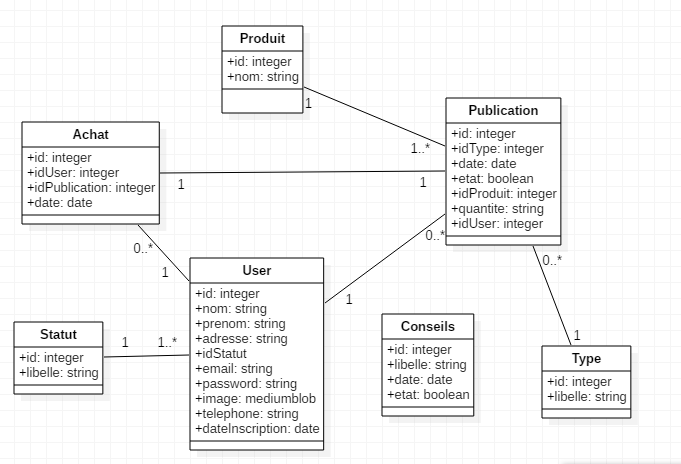
****

Figure 14 : Diagramme de classe de notre plateforme