Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
Université Virtuelle de Tunis

UVT
جامعة تونس الافتراضية
UNIVERSITÉ VIRTUELLE DE TUNIS

MEMOIRE

DE STAGE DE FIN D'ETUDES

Pour l'obtention du

«Mastère professionnel en Nouvelles Technologies des Télécommunications et Réseaux (N2TR)»

Présenté par :

Abdelmonem Talbi

Conception et développement d'un ERP propriétaire

Soutenu le :……

Devant le jury :

Président : Mr.(Mme.)……

Encadreur : Mr. Hassene Sedik

Rapporteur : Mr.(Mme.)……

Membre : Mr.(Mme.)……

Année Universitaire : 2016 / 2017

Dédicace

Je dédie ce travail à tous ceux qui ont fait de moi ce que je suis aujourd'hui, en témoignage de l'effort qu'ils ont déployé pour m'aider et qui a été toujours près de moi à me renforcer et me donner de l'espoir. Qu'ils trouvent ici l'expression de ma profonde gratitude et affection.

A mon épouse

Mon enfant Abderrahmen

&

A ma famille

Je dédie ce travail aussi à tous mes amis.

Remerciements

Je tiens tout d'abord à remercier Dieu le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce modeste travail.

En second lieu, Je tiens à remercier mes encadreurs

Mr. Hassene Sedik et Mr. Achref Athamna, Leurs précieux conseils et Leurs aides durant toute la période du travail.

Mes vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à mon projet en acceptant d'examiner mon travail et de l'enrichir par leurs propositions.

Enfin, Je tiens également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

SQL : Structured Query Language

PME : Petites et Moyennes Entreprises

PMI : Project Management Institute

ERP : **E**ntreprise **R**essources **P**lanning

PGI : Progiciel de Gestion Intégré

SAP: SSA global: Software Support Activity

SI: System Information

UML: Unified Modeling Language

Merise : Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise

JSF : Java Server Faces

JSP : Java Server Pages

CRM: Customer Relationship Management

GRC: Gestion Relation Client

MCD : Modèle Conceptuel des Données

API : Application Programming Interface

MVC: Model View Controller

IoC: Inversion of Control

EJB: Entity Java Bean

OMG: Object Management Group

DAO: Data Access Object

SGBD : Système de Gestion de Base de Données

EDI : Environnement de Développement Intégré

SVN : SubVersioN

VPN: Virtual Private Network

XML: Extensible Markup Language

AOP: Aspect Orient Programming

JDBC: Java Data Base Connection

ORM: Object Relationnal Mapping

JPA: Java Persistence Api

POM: Project Object Model

Introduction générale

Aujourd'hui, plusieurs entreprises de différentes spécialités sont prêtes à investir des sommes considérables dans l'implantation de nouvelles technologies logicielles. Ces entreprises ont pour finalité d'améliorer leurs services, d'accroitre leur agilité et flexibilité, de réduire leurs coûts, d'augmenter leur production ainsi que de faire face aux défis du marché. Cependant, la diversification et la croissance des activités au sein des entreprises ont fait grandir le besoin d'intégrer plusieurs technologies. La tâche de gérer efficacement toutes ces activités s'avère ainsi de plus en plus complexe et difficile.

Pour surmonter ses propres difficultés, une entreprise a besoin d'utiliser des outils optimisés qui s'adaptent à la réalisation de toutes les activités en offrant des fonctionnalités riches et utiles. Parmi ces outils, nous retrouvons les systèmes intégrés de gestion tels que les **ERP** (**E**ntreprise **R**essources **P**lanning) [1].

Les ERP sont des outils de gestion et d'analyse permettant d'optimiser la diffusion de l'information en interne, d'améliorer les processus de gestion et d'automatiser les activités et les tâches de l'entreprise. Selon Blain [2], l'utilisation des ERP a augmenté considérablement la réactivité, la productivité et la compétitivité des entreprises.

Le présent rapport est organisé en quatre chapitres. Le premier chapitre présente un aperçu sur les différents concepts théoriques d'un ERP, le contexte de notre travail qui est l'ERP et plus précisément l'intégration du module « gestion commerciale » dans notre ERP propriétaire appelé « DiagErp ».

Le second chapitre commence par une description du cadre du projet dans lequel nous allons définir notre champ d'étude. Par la suite, nous allons présenter l'architecture J2EE avec l'ensemble des standards ou *Frameworks* utilisés tels que Spring, Hibernate, JSF et Richfaces.

Dans le troisième chapitre, nous allons présenter les besoins fonctionnels sur lesquelles repose notre application Web. Par la suite, nous allons décrire ces besoins au moyen d'une modélisation en notation UML.

Le dernier chapitre abordera la mise en œuvre de l'application et exposera la partie expérimentale permettant de tester l'outil développé. Deux sections sont proposées pour mettre l'accent, d'une part sur le système informatique, et d'autre part, sur la vue applicative de l'outil développé. Une étude modulaire des fonctionnalités de l'application s'avère nécessaire pour définir précisément le futur périmètre qu'elle couvrira.

Finalement, nous dresserons le bilan de ce modeste travail ainsi que les perspectives d'avenir dans la conclusion de ce rapport.

Chapitre 1 : Les outils ERP

1. Introduction

L'acronyme ERP signifie « *Enterprise Resource Planning* » traduit en français par PGI « Progiciel de Gestion Intégré ». ERP est le terme le plus couramment utilisé.

Emanant d'un concepteur unique, un ERP est un progiciel qui permet de gérer l'ensemble des processus d'une entreprise intégrant l'ensemble de ses fonctions comme la gestion des ressources humaines, la gestion financière et comptable, l'aide à la décision, la vente, la distribution, l'approvisionnement, la production ou encore le e-commerce.

Dans ce premier chapitre, nous allons commencer par une présentation générale des ERP, et la description de leur architecture modulaire pour arriver enfin à une comparaison entre les deux types d'ERP, propriétaire et openSource.

2. Définition

Dans cette section, nous présentons les différentes définitions du terme ERP mentionnées par plusieurs auteurs, chacun selon son domaine de recherche :

Selon Caillaud [3], « Un système ERP est un progiciel qui vise à couvrir et optimiser la totalité des fonctions et des processus de gestion d'une organisation. Il s'appuie sur une couche « standard » pour traiter les besoins génériques et répond aux besoins spécifiques par des paramétrages. Il peut aussi fonctionner indifféremment sur plusieurs serveurs de données, systèmes d'exploitation et SGBD ».

Selon GOC [4] « les ERP sont des progiciels conçus pour la gestion des entreprises, composés de plusieurs modules entourant différents secteurs fonctionnels comme : la planification, la fabrication, les ventes, la distribution, la comptabilité financières, la gestion des ressources humaines, la gestion des projets, la gestion des stocks, le service et entretien, le transport et le e-business ».

SUPINFO [5] quant à lui énonce qu'il est nécessaire de « paramétrer de manière très poussée chacun des modules fonctionnels pour répondre aux besoins spécifiques de l'entreprise ».

À partir de ces définitions, nous pouvons déduire qu'un ERP est un progiciel permettant de gérer l'ensemble des processus de l'entreprise en intégrant plusieurs de ses fonctions telles que: la gestion des ressources humaines, la gestion financière et comptable, la

vente et la production, l'achat et l'approvisionnement tout en partageant une base de données commune.

D'autre part, il est à noter que le terme « ERP » est une appellation qui reflète plusieurs dénominations telles que: progiciel, progiciel intégré, progiciel applicatif, progiciel applicatif intégré, progiciel de gestion, progiciel de gestion intégré.

3. Utilité des ERP

3.1. Pour qui

Les ERP ont été principalement destinés aux grandes entreprises multinationales. Cependant, le marché des ERP tend à se démocratiser vers les PME selon le PMI [6.]. Certains éditeurs conçoivent un ERP uniquement pour ce type de structure. Par ailleurs, il existe des ERP open source avec le moindre coût, puisqu'il n'y a pas de coût de licence (ils sont gratuits). En revanche, il faut s'assurer de leur qualité en termes de performance. D'autre part, pour le calcul du coût d'acquisition total d'un ERP par un éditeur, il est nécessaire d'inclure les frais de maintenance et l'assistance technique.

3.2. Pourquoi

Concrètement, les avantages de la mise en place d'un ERP sont les suivants :

- L'intégrité et l'unicité du SI, c'est-à-dire qu'un ERP permet une logique et une ergonomie unique à travers sa base de données. Elle est unique au sens « logique ». Ceci se traduit par le fait qu'il peut exister plusieurs bases de données « physiques » mais celles-ci respectent la même structure. En bref, un ERP permet d'éviter la redondance de l'information entre différents SI de l'entreprise.
- L'utilisateur a la possibilité de récupérer des données de manière immédiate, ou encore de les enregistrer.
- Les mises à jour dans la base de données sont effectuées en temps réel et propagées aux modules concernés.
- Un ERP est un outil multilingue et multidevise. Ainsi il peut s'adapter au marché mondial, en particulier aux multinationales.
- Un ERP détient d'une synchronisation des traitements et d'une optimisation des processus de gestion, étant donné qu'il n'y a pas d'interface entre les modules. De même, la maintenance corrective est simplifiée puisqu'elle est assurée directement par l'éditeur et non pas par le service informatique de l'entreprise [6]. Néanmoins,

ce service garde sous sa responsabilité la maintenance adaptive (amélioration des fonctionnalités, évolution des règles de gestion, etc.) et préventive.

- Un ERP permet de maîtriser les stocks pour la plupart des entreprises, car les stocks coûtent relativement chers [1].

Par conséquent, les ERP gèrent et prennent en charge plusieurs périodes (pour les exercices comptables, par exemple), plusieurs devises, plusieurs langues pour les utilisateurs et clients, plusieurs législations, plusieurs axes d'analyse en informatique décisionnelle. Mais son implantation comporte plusieurs risques : des risques organisationnels (le progiciel et l'organisation de l'entreprise doivent cohabiter), de la mise en œuvre (au niveau formation de l'utilisateur), fonctionnels (fonctions offertes par le progiciel par rapport aux fonctions attendues), techniques, contractuels entre l'éditeur et l'entreprise ainsi que des risques économiques du fait de l'investissement.

4. Architecture technique

Le déploiement d'un ERP se fait, la plupart du temps, selon une architecture client/serveur comme le décrit le schéma de la figure 1.1.

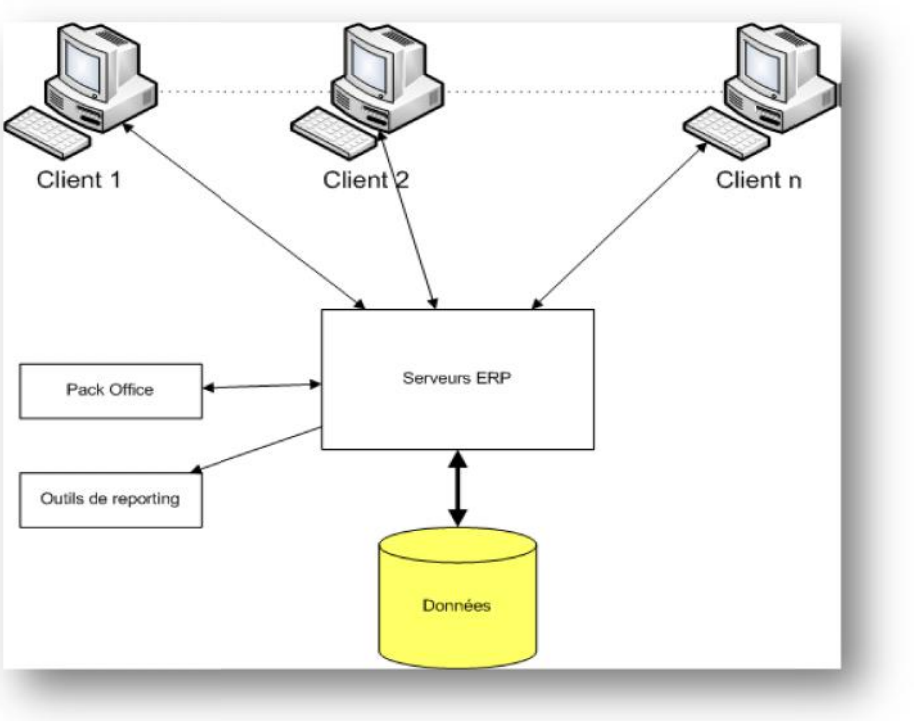


Figure 1-1 : ***Architecture technique d'un système ERP***

La figure 1.1 nous indique que la majorité des ERP se trouvent sur le serveur. Les serveurs ERP sont couplés à une base de données. De plus, les ERP sont compatibles pack Office, particulièrement, pour Powerpoint et Excel. En effet, le premier étant utile pour personnaliser les bureaux ERP en fonction de l'entreprise et le second pour effectuer les imports/exports de données. Enfin, les ERP sont aussi compatibles avec des outils de *reporting* (CrystalReport en général).

5. Architecture modulaire

Avant de découvrir en quoi une architecture modulaire consiste ainsi que ses modules, nous allons voir les principaux acteurs du marché des ERP (au niveau mondial).

5.1. Les principaux éditeurs d'ERP

Il y a deux types d'éditeurs d'ERP [7]: les ERP propriétaires, édités par des sociétés, ce qui implique l'achat d'une licence, et les ERP open source qui sont « gratuits » mais, il faut inclure dans le calcul du coût d'acquisition total, les frais de maintenance, de l'assistance technique et de la formation.

5.1.1. Les principaux ERP propriétaires

Parmi les principaux ERP propriétaires qui existent sur le marché, il y a les éditeurs suivants :

- SAP
- Oracle-PeopleSoft
- SAGE ADONIX
- Microsoft
- SSA Global

Comme toute autre ERP, les ERP propriétaires [8] présentent des avantages et des inconvénients (Tableau 1.1).

Tableau 1-1 : *Avantages et inconvénients d'un ERP propriétaire*

Avantages	Inconvénients
Fiabilité	Dépendance éditeur
Pérennité	Coûts
Assistance, support	Mise en œuvre
Puissance	
Fonctionnalités	

Des études annuelles menées par des cabinets de conseil permettent d'évaluer les parts de marché des grands éditeurs d'ERP propriétaires au niveau mondial [9].

- ***Le marché mondial des ERP***

Le schéma de la figure 2 nous présente la répartition des parts de marché des principaux ERP au niveau mondial [9]. SAP domine avec 40% des parts de marché.

Oracle qui avait racheté Peoplesoft détenait 22% des parts de marché. Sage était bien placé en particulier avec une clientèle de PME.

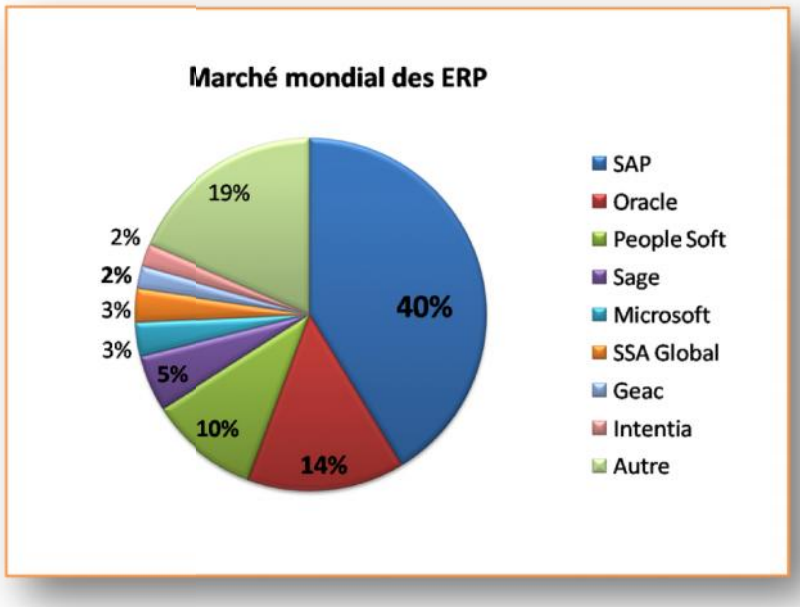


Figure 1-2 : ***répartition de marché mondial des*** *ERP*

- ***Le marché français des ERP***

Le schéma de la figure 1-2 présente la répartition des parts de marché des principaux ERP en France [9]. SAP domine aussi le marché, avec 39% des parts de marché.

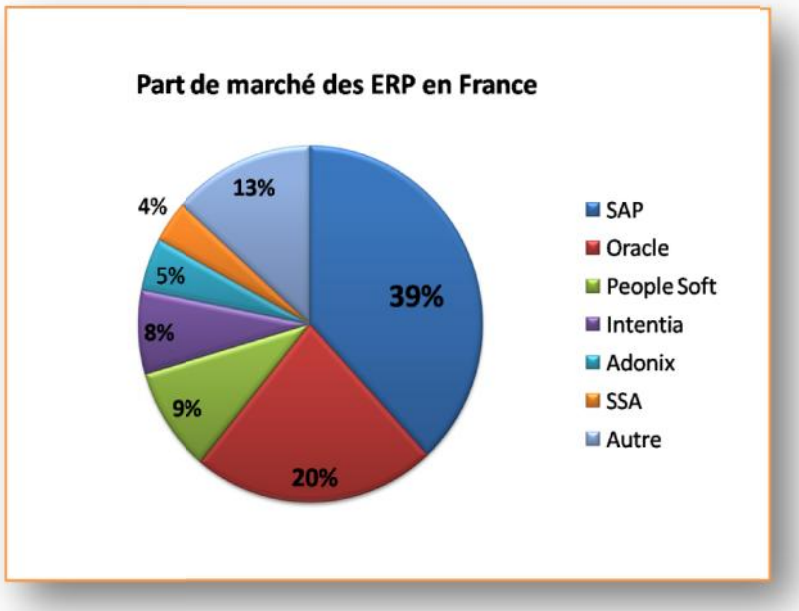


Figure 1-3 : *Répartition des marchés ERP propriétaire en France*

- ***Marché ERP en terme d'argent***

Après une baisse relative, le marché des ERP repart à la hausse. Il y a au moins deux raisons:

- le besoin de la cohérence des systèmes d'information n'a jamais été si grand,
- les ERP ont beaucoup progressé et sont désormais, plus souples, plus facile à installer et plus performants.

D'après les prévisions de Forrester [10], le marché global des ERP atteindra 45,5 milliards de dollars en 2018. Ce marché qui est passé de 40,6 milliards en 2016 à 43 milliards en 2017 (figure 1-3), devrait atteindre 50,3 milliards en 2019. Cette croissance devrait inciter les entreprises à reconsidérer leurs engagements actuels et leurs plans en matière d'ERP.

Oracle et SAP demeurent les deux concurrents dominants (figure 1.4). Mais, les nouveaux spécialistes de la consolidation et de l'industrie, et les nouvelles technologies qui améliorent la souplesse des process business vont modifier l'offre du marché.

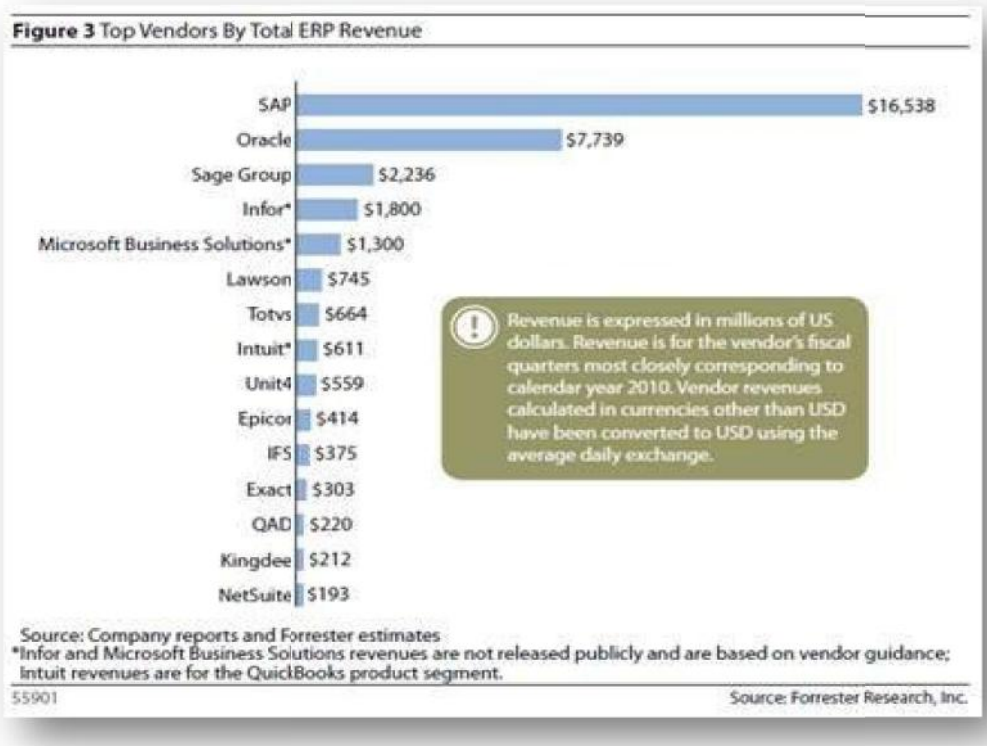


Figure 1-4 : *Marché des ERP en million de dollars [10]*

5.1.2. Les principaux ERP openSource

Un ERP Open Source est différent d'un logiciel ERP propriétaire, non pas en ce qui concerne les fonctionnalités disponibles, mais plutôt à propos de la licence du produit et sa personnalisation. Comme exemple, nous retrouvons :

- **Compiere**
- **Aria**
- **ERP5**
- **OFBiz**
- **Openbravo**

Les ERP openSource présentent aussi certains avantages et inconvénients tels qu'illustrés dans le tableau 1-2 [12] :

Tableau 1-2 : *Avantages et inconvénients d'un ERP open Source*

Avantages	Inconvénients
Agilité, flexibilité	Assistance, support
Spécificité	Puissance
Coûts	Pérennité
Indépendance	
Mise en œuvre	

5.2. Les modules généraux d'un ERP

Un ERP est un ensemble dont tous les modules fonctionnent les unes avec les autres d'où l'ergonomie et l'unicité des informations et ainsi la cohérence du SI.

Un ERP est modulaire dans le sens qu'il est possible d'avoir une ou plusieurs applications, en même temps. Dans un ERP, les applications sont modulaires, et par conséquent, la compatibilité entre les modules est assurée. Ces modules s'imbriquent comme des blocs de Lego et fonctionnent ensemble (il n'est pas nécessaire d'effectuer une vérification de compatibilité). La figure 1.5 illustre un exemple d'architecture modulaire qui tend à représenter tous les ERP (propriétaire et openSource). Cette architecture modulaire intègre plusieurs modules portant sur les principales fonctions d'une entreprise telles que : le module finance, logistique et e-commerce.

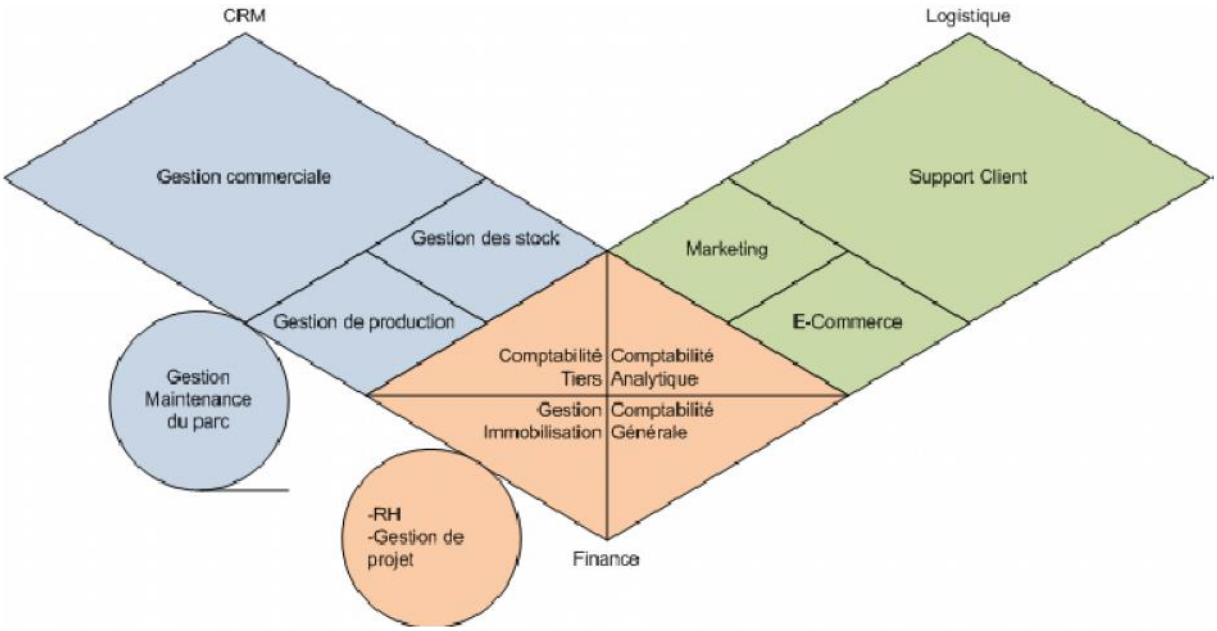


Figure 1-5 : ***Architecture modulaire des ERP***

5.2.1. Le module finance

Le module finance se divise en cinq sous-modules. Il collecte les données pertinentes (en particulier, les données comptables de l'entreprise) pour établir des comptes annuels sur un plan international. Ce module effectue le contrôle de gestion et les prévisions selon les objectifs de l'entreprise. Il permet également d'effectuer la comptabilité tiers, analytique et générale, de gérer les immobilisations (gestion des investissements), les ressources humaines, telles que l'administration du personnel, des frais de déplacement et du temps de travail ainsi que la gestion de paie et les besoins en postes.

Ce module est relativement complexe, car, il est nécessaire d'avoir des connaissances approfondies en comptabilité.

5.2.2. Le module logistique

Le module logistique, appelé encore Negoce, est le module le plus convoité par les entreprises, car il permet de gérer tout ce qui se rapporte aux ventes/achats, en particulier la gestion des stocks qui représente des coûts élevés pour la plupart des entreprises [10].

Ce module gère les commandes clients et les livraisons. Il assure une liaison directe entre le compte résultats et le système de production. Il permet l'optimisation des processus de *workflow*, la gestion précise des stocks, des contrôles de la qualité et des factures. Suite au contrôle de la qualité, des mesures correctives sont déclenchées au fur et à mesure de

l'avancement du projet. Enfin, il est possible de planifier, gérer et suivre la maintenance du matériel.

5.2.3. Le module e-commerce

Ce module permet le commerce électronique ou la vente en ligne, communément appelé *Customer Relationship Management* (CRM) ou encore Gestion Relation Client (GRC) [11]. Il permet d'effectuer les statistiques (sous forme de requêtes) sur tous types de données (relatifs aux clients, aux vendeurs, à la production, aux fournisseurs, etc.).

6. Comparaison entre un ERP propriétaire et un ERP open Source

Le tableau suivant montre une comparaison entre ERP open Source et un ERP propriétaire. Cette comparaison est effectuée par rapport aux objectifs de l'entreprise (tableau 1.3).

Tableau 1-3 : *Comparaison entre un ERP propriétaire et celui openSource*

ERP propriétaire	**ERP openSource**
Si l'entreprise dépasse des milliers de transactions par mois et plusieurs dizaines d'utilisateurs faisant des requêtes simultanées	Croissance incompatible avec le système de tarification des ERP commerciaux
Traitements métiers spécifiques	Compétitivité native
Grande société	

7. Conclusion

Dans ce premier chapitre, nous avons pu collecter de l'information sur les **ERP**. Nous avons commencé par présenter les principes et les concepts de base des ERP.

A travers cette présentation, nous avons pu constater la place prépondérante que prennent actuellement les ERP au cœur des différents systèmes des organisations. En effet, l'intérêt des ERP semble être double. D'une part, nous pouvons retenir un indiscutable intérêt technique dû principalement à l'intégration des données de l'entreprise qui permet un gain de temps et d'argent considérable. D'autre part, ce sont des progiciels de gestion prêts à implanter couvrant l'ensemble des processus de l'organisation de l'entreprise.

Dans le chapitre 2, nous allons présenter notre champ d'étude pour spécifier nos besoins ainsi que de définir le choix de la technologie de développement.

Chapitre 2 : Étude préalable

1. Introduction

La mise en œuvre d'un système d'information doit passer par l'étape de l'étude préalable qui permet d'obtenir des spécifications fonctionnelles du futur système. Une première section est consacrée à la définition du champ de l'étude ou du cadre de notre projet. Ensuite, dans la deuxième section, nous allons présenter l'architecture à utiliser en termes de processus de développement ainsi que le modèle de développement et les standards utilisés.

2. Champ d'étude

Notre projet consiste à concevoir et développer un module de gestion commerciale pour les entreprises commerciales et son intégration dans un ERP « DiagErp » tout en respectant les normes définies par la société telle que l'internationalisation de l'application et la simplicité de l'utilisation. La gestion commerciale se base essentiellement sur un module de gestion des achats qui permet de gérer les transactions d'achat et écritures comptables associées. Ainsi, elle assure la gestion de la comptabilité, les approvisionnements selon des politiques à paramétrer et selon le calcul des besoins déterminés par la gestion de production. Par ailleurs, la gestion des articles fait partie de la gestion des stocks.

2.1 Organisme d'accueil

GROW-TIC est une société Franco-Tunisienne (filiale de la société DIAG) classée dans la catégorie de société offshore Elle est spécialisée dans le développement des applications de gestion web basées sur la technologie J2EE en utilisant des *frameworks* récents tels que Spring, hibernate, Jsf, maven, etc.

GROW-TIC est encore dans ses premiers pas, elle est créée en 2016. Son domaine d'activité ne s'exerce que sur le marché européen. L'équipe GROW-TIC se compose d'un directeur et six ingénieurs hautement qualifiés. Cette équipe a pour objectif d'optimiser ses processus internes, fournir des services de bonne qualité pour les clients ainsi que d'améliorer la productivité et la rentabilité de l'entreprise.

2.2 Objectifs à atteindre

Les principaux objectifs à atteindre concernant l'application DiagErp sont :

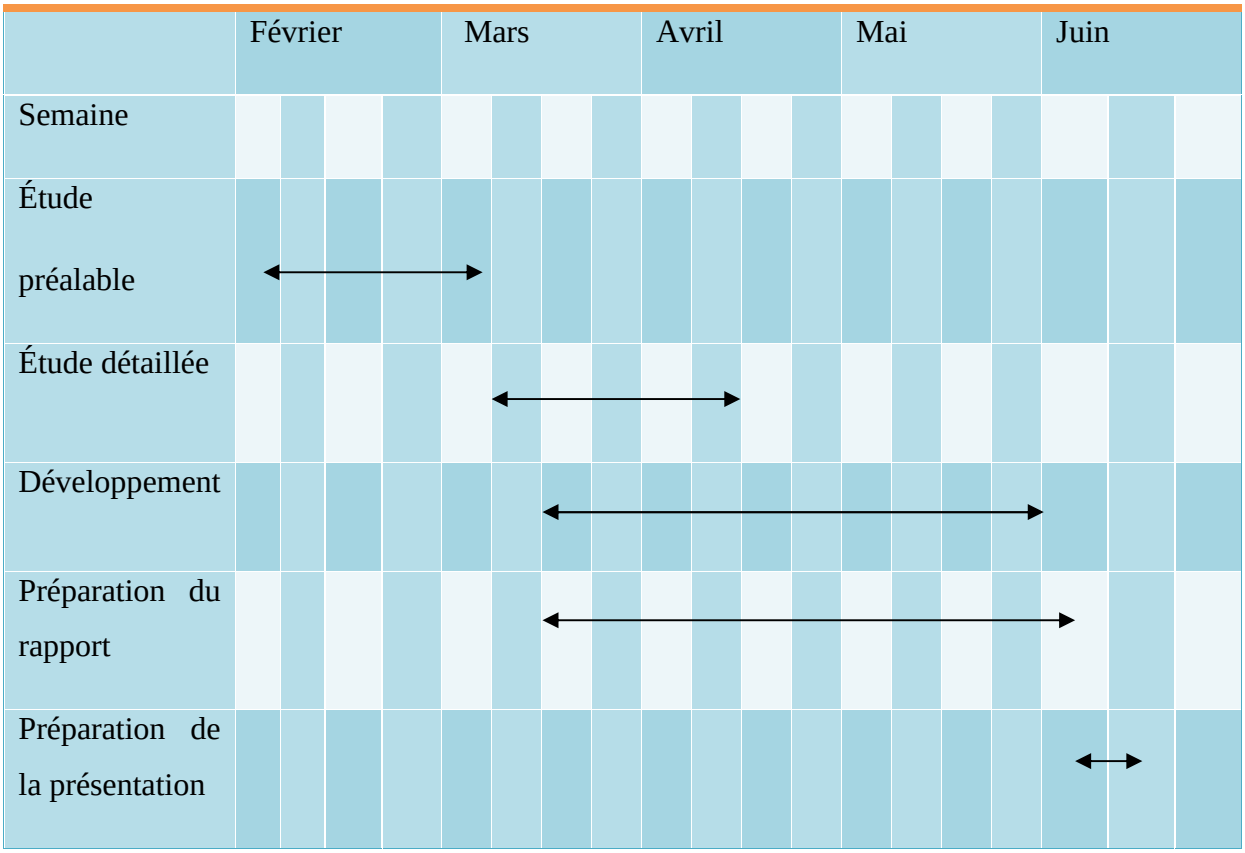
- Réalisation des opérations de base telles que la saisie et l'interrogation des commandes fournisseurs ;
- Edition de commandes fournisseurs ;
- Réception des commandes fournisseurs ;

- Gestion des articles ;
- Gestion de stock ;
- Gestion des paramètres comptables.

2.3 Planning prévisionnel

Le tableau 2.1 présente notre planning prévisionnel.

Tableau 2-1 : *Planning de projet*



3. Etude de l'existant

3.1 Analyse de l'existant

Basée sur le langage Cobol, l'application en cours d'utilisation présente une facilité d'utilisation remarquable et une stabilité non négligeable. Cependant, le mode de navigation écran par écran ainsi que la présentation des données fournissent une expérience utilisateur assez rudimentaire. Pour ce qui est de la persistance des données, l'application utilise la notion de fichier et non pas la notion de base de donnée relationnelle.

L'application est déployée chez le client sur un serveur Linux. Dans certains cas, la maintenance peut se faire à distance.

3.2 Critique de l'existant

Dans un contexte où les applications Web montrent des évolutions au niveau des versions assez importantes, il serait impératif d'adopter un système de développement qui permet non seulement de gagner du temps mais aussi de prévoir une maintenance et une évolutivité assez aisée. Par conséquent, il y aura un gain en termes de temps et de la main d'œuvre employée pour la réalisation de tels projets. Ce qui n'est pas le cas pour l'application existante. Pour l'application en cours de production, il serait préférable d'envisager un modèle économique qui permet de réaliser une maintenance à moindre coût, d'une part et de minimiser les coûts d'hébergement en suivant par exemple le modèle de l'informatique sur les nuages, d'autre part.

4. Etude technologique

4.1 Plateforme J2EE

J2EE est une spécification Java de Sun destinée particulièrement aux applications d'entreprise [14]. J2EE est l'une des solutions pour répondre aux besoins exprimés dans la section 3. J2EE spécifie à la fois :

- une infrastructure de gestion des applications permettant de les exécuter,
- un ensemble d'API (*Application Programming Interface*) pour concevoir les applications.

La plateforme J2EE est un environnement d'exécution d'applications menée d'interfaces qui représentent des services existants. Cet environnement d'exécution fournit des services d'infrastructure (transaction, persistance, annuaire, etc.) qui étaient traditionnellement développés dans les codes applicatifs.

J2EE ne spécifie ni la nature ni la structure de l'environnement d'exécution. Mais plutôt, il introduit un conteneur, et via les API de J2EE, il élabore un contrat entre les conteneurs et les applications. En outre, les API présentent l'avantage d'être faciles à utiliser. Elles permettent de cacher la complexité d'accès aux ressources et par conséquent de gagner considérablement du temps de telle sorte que les développeurs puissent consacrer plus de temps aux aspects « métier ». Il existe deux types de services de l'API : des services

d'infrastructure et des services de communication. Dans le tableau 2-2, nous exposons quelques services [14] :

Tableau 2-2 : *Les différentes API J2EE*

Nom de l'API	Description
JDBC – Java Data Base Connectivity	API d'accès aux bases de données. Son utilisation diminue le nombre de lignes de code à écrire. De plus, les accès peuvent être optimisés à l'aide des pools de connexions fournis par les serveurs d'application.
JNDI	API d'accès aux services de nommage et aux annuaires d'entreprises (DNS, LDAP, etc.)
JTA/ JTS: Java Transaction Api/ Java Transaction Services	API définissant des interfaces standards avec un gestionnaire de transactions.
JCA (J2EEConnector Architecture)	API de connexion au Système d'Information de l'entreprise (ERP, etc.).
JMX (Java Management eXtension)	API permettant de développer des applications WEB de supervision d'applications.
JAAS (Java Authentification and Authorization Service)	API de gestion de l'authentification et les droits d'accès.
RMI (Remote Methode Invocation)	API permettant la communication synchrone entre objets.
Web Services	Permettent de partager un ensemble de méthodes qui pourront être appelées à distance.
JMS (Java Message Service)	API fournissant les fonctionnalités de communication asynchrone.

En simplifiant, nous pouvons dire que J2EE est une collection de composants, de conteneurs et de services. Ces conteneurs permettent de créer et de déployer des applications distribuées au sein d'une architecture standardisée.

❖ ***Côté client***

Un client J2EE peut être une application console (texte seulement) écrite en Java, ou une application dotée d'une interface graphique développée en Swing. Ce type de client est appelé client lourd, en raison de la quantité importante du code qu'il met en œuvre [15]. Un client J2EE peut également être conçu pour être utilisé à partir du Web. Ce type de client fonctionne à l'intérieur d'un navigateur Web. La plus grande partie du travail est reportée sur le serveur et le client ne comporte que très peu de code. Pour cette raison, nous parlons de client léger. Un client léger peut être une simple interface HTML, une page contenant des scripts JavaScript, ou encore une applet Java si une interface un peu plus riche est nécessaire (figure 2.1).

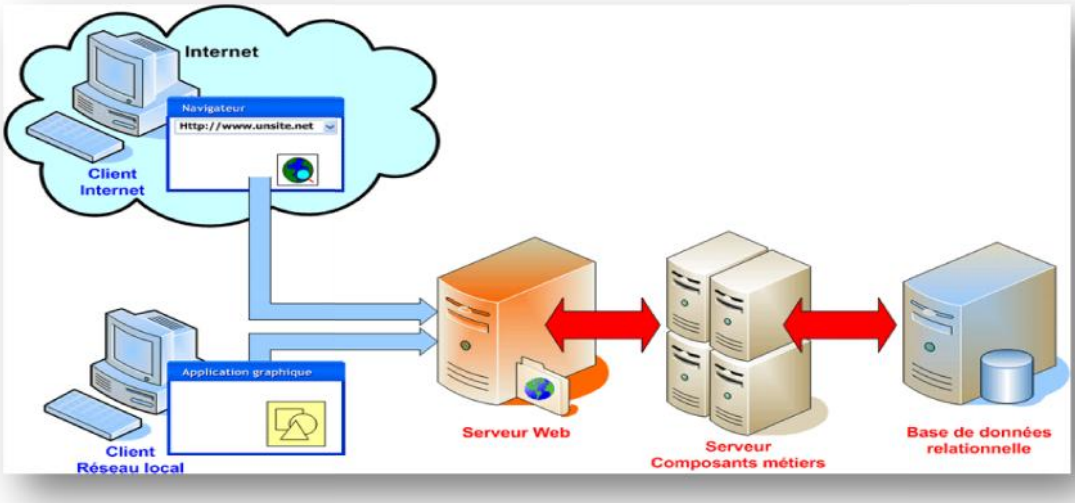


Figure 2-1 : *Comportement coté client*

❖ ***Côté serveur***

Les composants déployés sur le serveur peuvent être classés en deux groupes. Les composants Web sont réalisés à l'aide des Servlets ou de Java Server Pages (JSP). Les composants métiers, dans le contexte J2EE, sont des EJB (figure 2.2) [13].

J2EE et la plate-forme Java disposent de conteneurs pour les composants Web et les composants métiers. Ces conteneurs possèdent des interfaces leur permettant de communiquer avec les composants qu'ils hébergent. Le choix des *frameworks* de développement repose sur Spring, Hibernate, JSF vue leur impressionnante réputation fonctionnelle tout en respectant les règles d'art que pose le modèle MVC (*Model View Controller*).

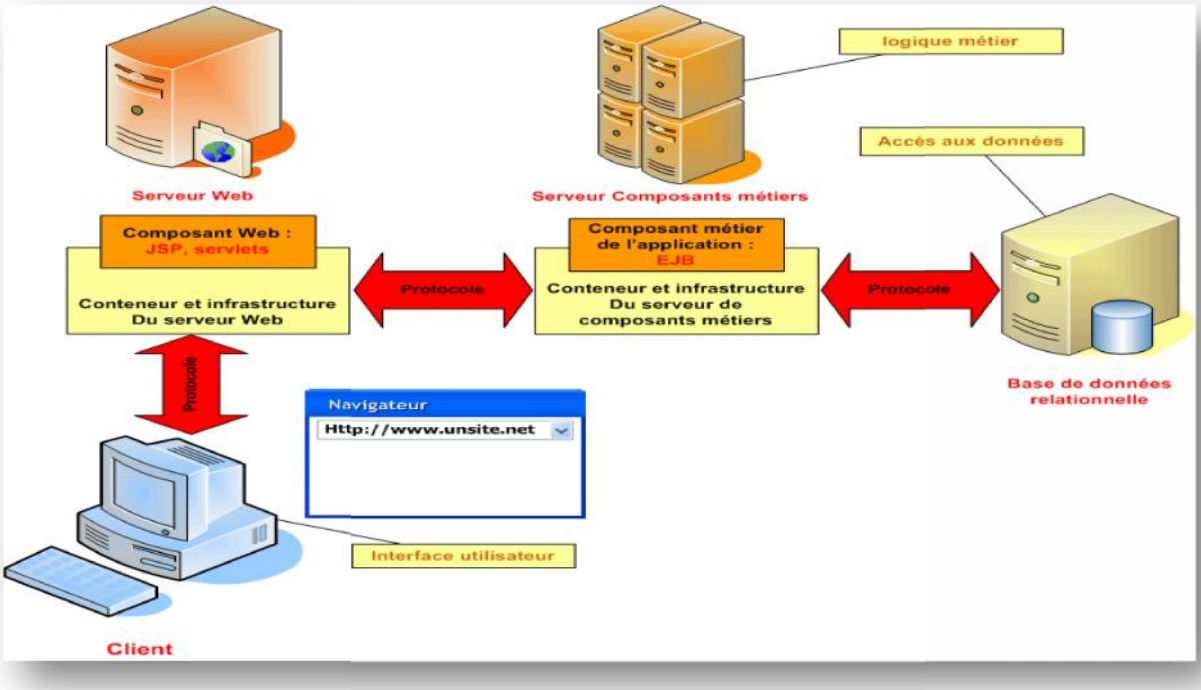


Figure 2-2 : *Comportement côté serveur*

4.1.1 Spring

Spring est un conteneur d'objets Java, il est conçu pour simplifier le développement d'applications 3-tiers J2EE [16]. Il est réputé comme le *framework* le plus utilisé des *frameworks* d'application Java/J2EE. Les apports consistent à :

- ✓ rendre J2EE plus facile à utiliser.
- ✓ fournir la meilleure solution IoC (Inversion of control) : Il prend en charge la création d'objets et la mise en relation entre eux par l'intermédiaire d'un fichier de configuration. Ce fichier décrit les objets à fabriquer et les relations de dépendances entre eux.
- ✓ fournir des couches d'abstractions : il y a la couche de services obéissant à la solution IoC. Cette couche restreint l'accès direct à la base de données à partir de la couche métier.
- ✓ fournir la portabilité sur tous les serveurs d'application : Spring peut être implémenté sous Weblogic, Apache Tomcat, WebSphere, JBoss, etc.

4.1.2 Hibernate

Hibernate est un Framework de projection Objet/Relationnel pour les applications JAVA. La projection Objet/Relationnel consiste à décrire une correspondance entre un

schéma de base de données et un modèle de classes pour s'assurer de la persistance de l'état des objets. Pour cela, nous utilisons des fichiers de projection XML [17].

Il permet de créer une couche d'accès aux données (Data Access Object) plus maintenable, plus performante qu'une couche d'accès aux données « classiques » reposant sur l'API JDBC.

Hibernate génère automatiquement le code SQL. Il fournit plusieurs stratégies pour interroger la base de données par des requêtes SQL ou le langage HQL avec des options de recherche et de mise en cache sophistiquées.

4.1.3 JSF

Java Server Faces, JSF est une spécification qui permet de créer des pages web dynamiques. Ces pages peuvent être par exemple des pages xhtml, jspx, jsf.

Développé à travers le « Java Community Process » sous JSR-314 [20], la technologie Java Server Faces établit un standard pour la construction d'interfaces utilisateurs coté serveur. Avec la contribution d'un groupe d'experts, les api JSF sont conceptualisés de telle sorte qu'il est possible de tout développer autour des outils qui font que le développement des applications web soit encore plus facile [18].

JSF permet de faciliter le développement web en assurant par exemple la gestion de session ou la définition des règles de navigation entre les pages. JSF permet aussi l'utilisation du modèle pattern MVC (Model Vue Controller), ce qui permet de séparer les tâches entre développeur et concepteur. Il intervient aussi dans le processus de validation et conversion des données de la requête à traiter. Les différents aspects de JSF seront détaillés dans le chapitre 4.

Le Design Pattern MVC fournit une solution au problème que pose le développement d'une application classique à savoir la séparation entre la présentation et la logique métier. Ce qui facilite, pour une même logique métier, plusieurs présentations sur des plateformes différentes. MVC propose un découplement entre la présentation, le contrôle et le modèle (figure 2-3).

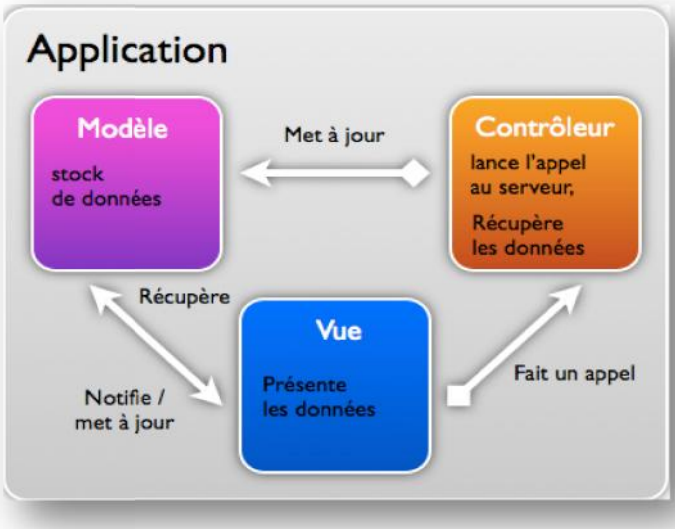


Figure 2-3 : ***Modèle MVC***

Plus récemment, le modèle MVC a été recommandé pour la plate-forme J2EE de Sun. Ainsi, il a gagné la popularité auprès des développeurs, quel que soit le langage utilisé. Ainsi, lors de sa connexion, le client envoie une requête à l'application [19]:

- ✓ la requête envoyée depuis la vue est analysée par le contrôleur,
- ✓ le contrôleur demande au modèle approprié d'effectuer les traitements,
- ✓ le contrôleur renvoie la vue adaptée, si le modèle ne l'a pas déjà fait.

➢ **Modèle** :

Le modèle représente les données et les règles métiers. C'est dans ce composant que s'effectuent les traitements liés au cœur du métier.

➢ **Vue** :

La partie Vue représente l'interface utilisateur. Elle n'effectue aucun traitement, elle se contente simplement d'afficher les données fournies par le modèle. Il peut y avoir plusieurs vues qui présentent les données d'un même modèle.

➢ **Contrôleur**

Le contrôleur se charge d'intercepter les requêtes de l'utilisateur, d'appeler le modèle puis de rediriger vers la vue adéquate. Il ne doit faire aucun traitement. Il ne fait que de l'interception et de la redirection.

4.1.4 Richfaces

Richfaces est une librairie de composants JSF pour le développement d'applications web riches (RIA) Rich Internet Application avec Ajax, elle est le résultat de la fusion de deux projets développés par exadel qui sont [20]:

- **Ajax4jsf** : crée originalement par la fondation sourceforge.net en 2005 sous le nom de Telamon, son concepteur a intégré ensuite la société Exadel qui a commercialisé le projet dans une première période avant de le rendre open source sur Java.net.
- **Richfaces** : C'est une librairie commerciale de composants JSF fournit par Exadel.

Ces deux projets sont passés dans le giron de JBoss en septembre 2007 pour former un projet open source nommé JBoss Richfaces.

5. Conclusion

A travers ce chapitre, nous avons pu faire un tour d'horizon sur le cadre général du projet. Nous avons mis l'accent sur notre objectif général qui se résume dans la gestion commerciale des entreprises, ainsi une description générale sur l'architecture utilisée en se focalisant sur les Frameworks est mise en évidence pour le traitement de nos objectifs.

Dans le prochain chapitre, nous allons spécifier d'une manière plus approfondie l'objectif à aboutir tout en présentant la conception de notre projet.

Chapitre 3 : Analyse et conception

1. Introduction

Lors de ce chapitre, nous allons identifier les diagrammes de classes, des cas d'utilisation et de séquences réalisés pour mettre en œuvre l'architecture proposée. La motivation fondamentale de la modélisation est de procéder à une démarche requise à la réalisation. Cette démarche vise à réduire la complexité du système étudié lors de la conception et d'organiser la réalisation du projet en définissant les modules et les étapes de la réalisation.

Désormais, le choix d'un langage de modélisation adéquat s'avère important pour mieux représenter le système à concevoir. Le choix du langage de modélisation dans le cadre de notre projet s'est porté sur UML vue qu'il est très proche des notions orientées objet et couvre tous les aspects et les vues de la conception d'un projet. Dans notre travail, nous adoptons une approche objet basée sur un outil de modélisation UML.

2. Phase d'analyse

Afin de comprendre nos objectifs par rapport à la cible de l'application à développer, nous nous sommes penchés sur les besoins de GROW-TIC (filiale de DIAG). Ainsi, nous veillons sur les aspects essentiels de leur métier et leurs attentes par rapport à la nouveauté technologique. Notre objectif est d'éclaircir les points importants que nous devrons respecter dans la phase de conception.

En effet, la mission de l'utilisateur de l'application à développer consiste à gérer la partie commerciale de l'entreprise qui englobe trois parties incluant : un module client, un module fournisseur et un module comptabilité. Ces trois modules sont présentés dans la figure 3.1. Il est à signaler qu'à ce niveau, notre projet porte seulement sur le module fournisseur.

Le module fournisseur consiste à définir le processus d'approvisionnement en matière première dès la naissance d'une demande d'achat jusqu'à la gestion comptable tout en passant par la saisie de la commande fournisseur et sa réception.

Le processus peut être déclenché suite à une alerte du stock ou à travers la réception d'une commande client dont la quantité demandée est supérieure à celle du stock existant.

La réception d'un bon de livraison entraine la saisie des factures afin de mettre à jour les plans comptables de la société et générer le paiement qui peut prendre plusieurs formes de règlement. La mise à jour de ces plans nécessite la saisie des écritures comptables.

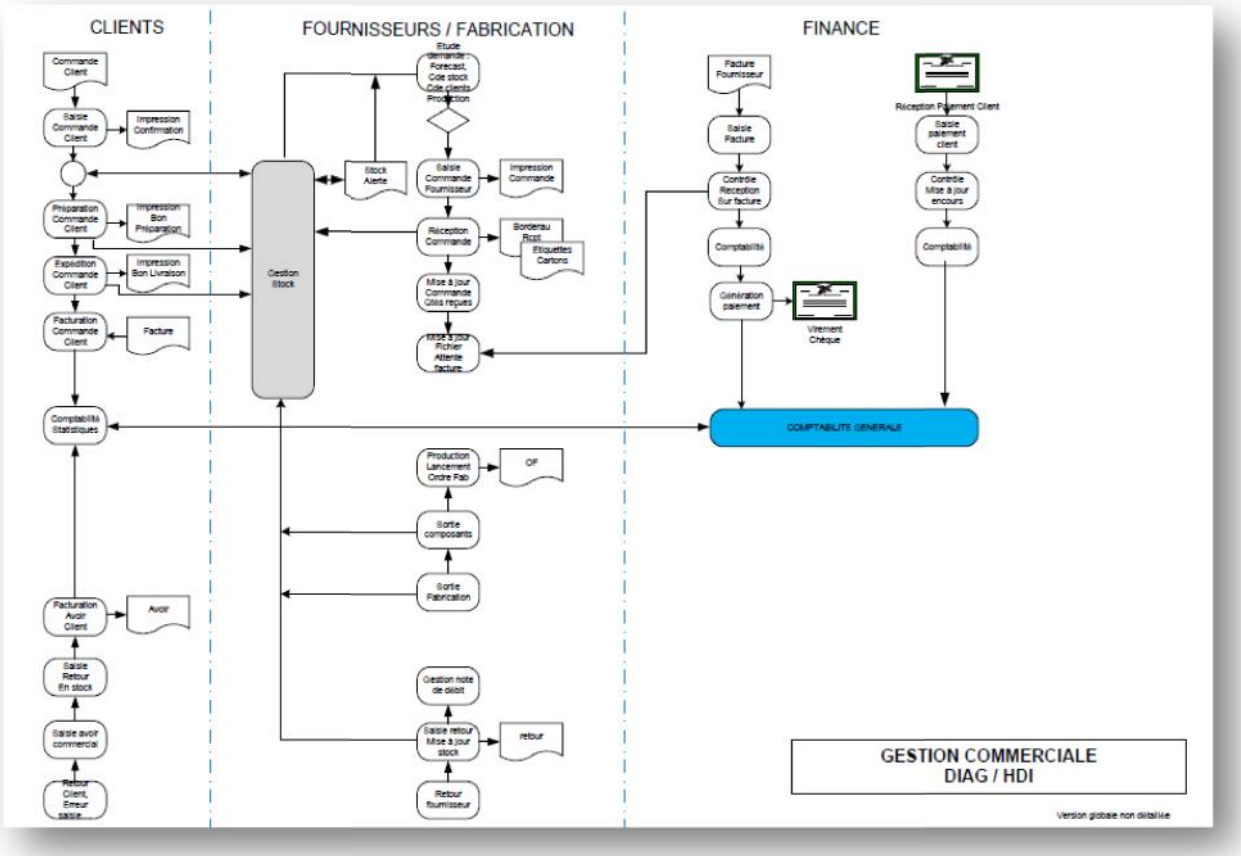


Figure 3-1 ***vision générale du projet***

3. Langage de modélisation UML

UML (*Unified Modeling Language*) est un standard ouvert contrôlé par l'OMG (*Object Management Group*), un consortium d'entreprises qui a été fondé pour construire des standards qui facilitent l'interopérabilité et plus spécifiquement, l'interopérabilité des systèmes orientés objet [21].

UML est issu de l'unification de nombreux langages de modélisation graphique orientée objet. Il unifie à la fois les notations et les concepts orientés objets.

En outre, pour modéliser notre projet, nous allons identifier trois diagrammes:

- ✓ Les diagrammes de cas d'utilisations représentent un intérêt pour l'analyse des besoins métier, ce qui nous permettra de démarrer l'analyse orientée objet et identifier les classes candidates.
- ✓ Les diagrammes de séquences qui sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation UML.

- ✓ Un diagramme de classes est une collection d'éléments de modélisations statiques (classes, paquetages, etc.), qui montre la structure d'un modèle. Les classes sont liées entre elles par des associations. Une association permet d'exprimer une connexion sémantique bidirectionnelle entre deux classes.

3.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est un diagramme UML utilisé pour donner une vision globale du comportement fonctionnel d'un système logiciel. Un cas d'utilisation représente une unité discrète d'interaction entre un utilisateur (humain ou machine) et un système. Plus précisément, un cas d'utilisation décrit une séquence d'actions réalisées par le système qui produit un résultat observable pour un acteur [21].

Les diagrammes présentés dans les figures illustrent les cas d'utilisations de notre application. Ils décrivent le comportement du système d'un point de vue utilisateur. Le premier diagramme (figure 3.2) représente le cas d'utilisation de la gestion des articles et des fournisseurs. En effet, après son authentification l'utilisateur est ramené à contrôler les articles existants dans les différents sites de stockage, par ajout, modification ou suppression. Il peut également effectuer les mêmes opérations sur les fournisseurs. Il peut asussi contacter un fournisseur par e-mail. Dans les deux cas, une édition de rapport sous format PDF est possible.

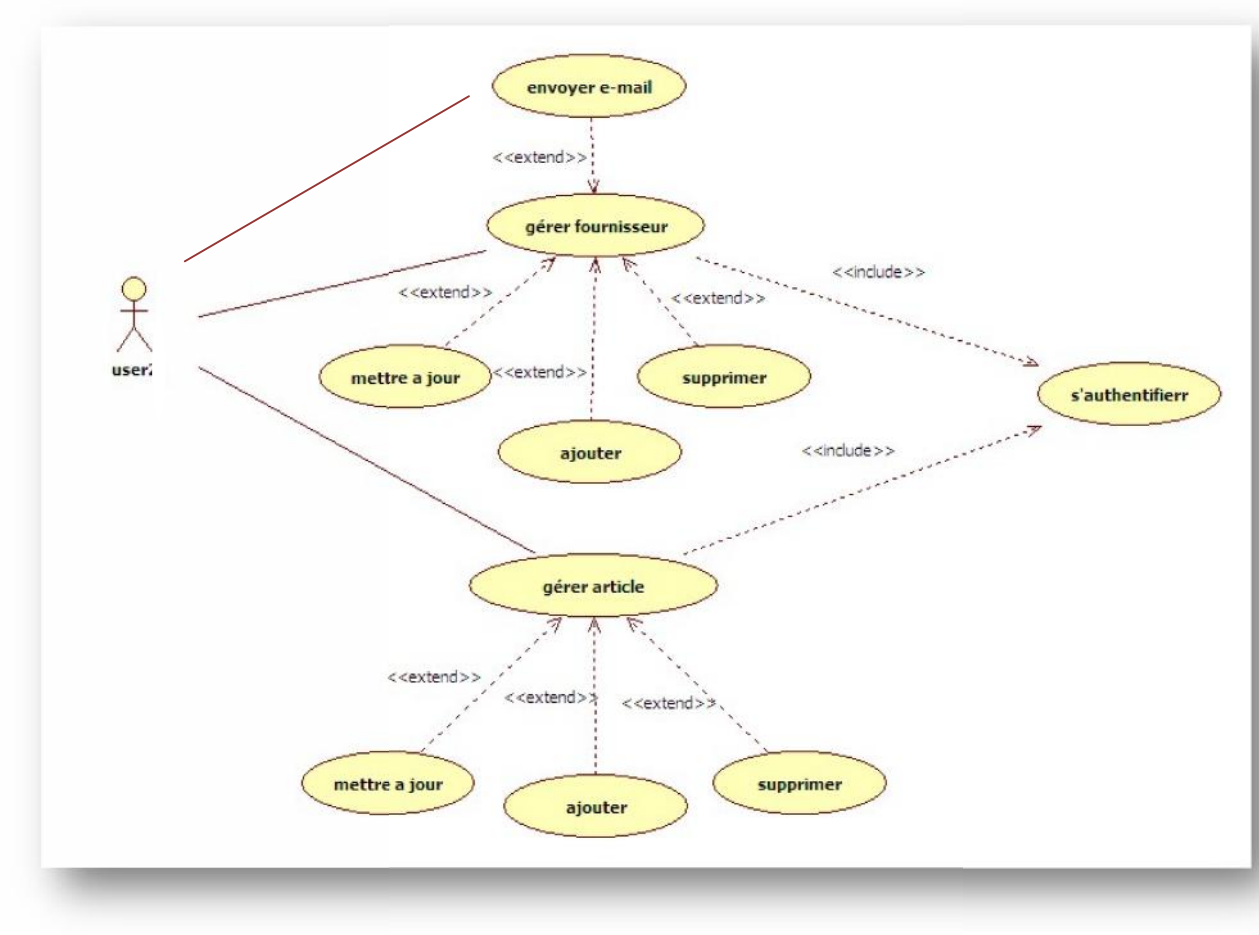


Figure 3-2 : *Cas d'utilisation gestion article et fournisseur*

La figure 3.3 représente le cas d'utilisation de saisie des commandes fournisseurs. Une commande est divisée en deux parties, un entête qui contient des informations générales et standards sur le fournisseur correspondant, et une partie de saisie des lignes de commande. Lors de saisie d'une ligne, l'utilisateur doit être informé s'il y a des commandes en cours de ce même article. Ainsi d'effectuer un contrôle sur la quantité minimale demandée qui peut être approvisionné par le fournisseur.

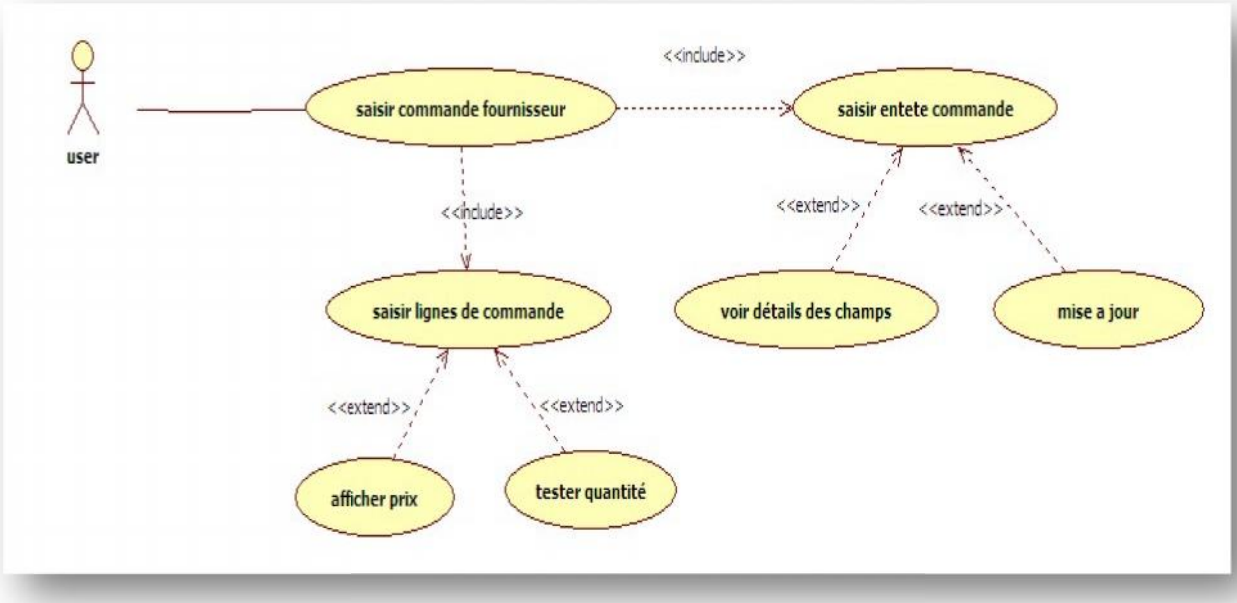


Figure 3-3 : *Cas d'utilisation saisie commande fournisseur*

Après avoir saisie une commande, l'utilisateur peut interroger les commandes du fournisseur existantes en effectuant les opérations de base telles que la mise à jour ou la suppression, ou effectuer une édition commande soit par extraction en fichier PDF soit par envoi direct via un email en utilisant l'adresse email de la table fournisseur.

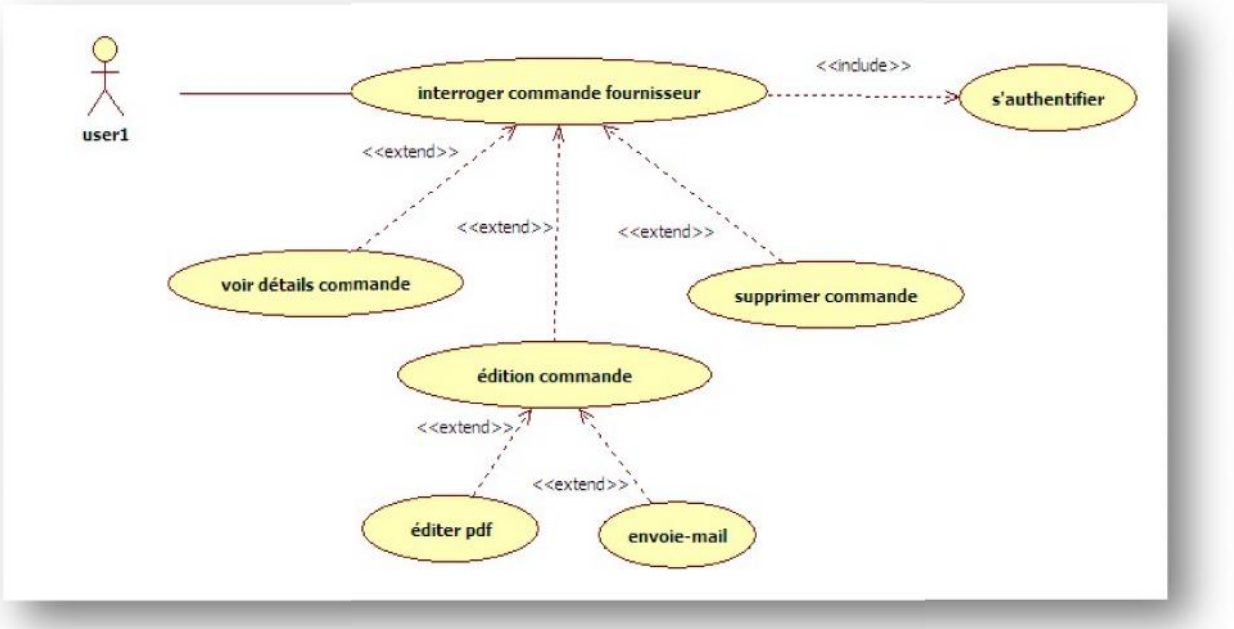


Figure 3-4 : *Cas d'utilisation interrogation commande fournisseur*

Enfin, la figure 3.5 représente le cas d'utilisation de la réception d'un bon de livraison. La réception d'une commande fournisseur passe par plusieurs étapes. En effet, pour pouvoir payer la commande reçue, l'utilisateur doit régler une facture qui engendre, par la suite, la saisie des écritures comptables. Enfin, l'utilisateur doit mettre à jour le stock approvisionné de l'entreprise.

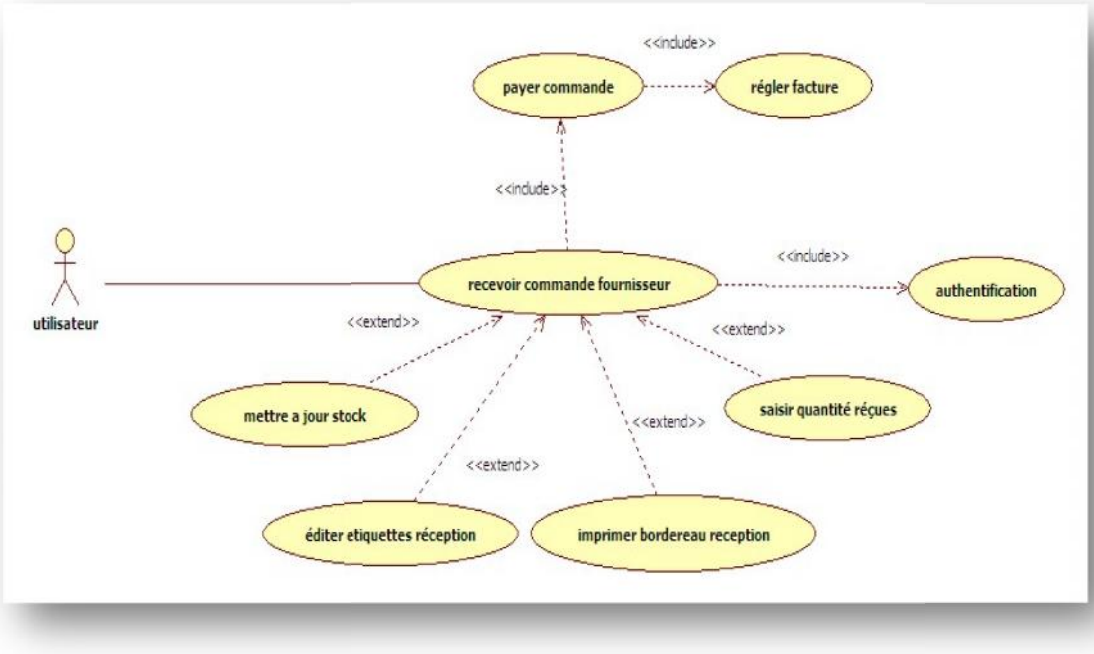


Figure 3-5 : *Cas d'utilisation réception commande fournisseur*

Dans cette section, nous allons décrire sous forme textuelle quelques cas d'utilisation présentés dans les figures …

3.1.1 Authentification

- Scénario nominal :

  Titre : Authentification de l'utilisateur

  Acteur principal : utilisateur

  Pré-condition : Connexion Internet établie

  Post-condition : Ouverture du compte

Tableau 3-1 : *Scénario du cas d'utilisation authentification*

Action acteur	Réactions du système
1. Le planificateur se connecte au système.	2. Le système demande à l'utilisateur de saisir son identifiant et son mot de passe.
3. L'utilisateur introduit son identifiant et son mot de passe.	4. Le système vérifie l'identifiant et le mot de passe. 5. Le système affiche la page d'accueil

- Scénario alternatif :

Ce scénario représente l'échec d'authentification. Le scénario alternatif démarre après l'étape 4. Le système indique à l'utilisateur l'échec d'authentification et lui demande de réintroduire son identifiant et son mot de passe.

3.1.2 Ajout article

- Scénario nominal :

Titre : Ajout article

Acteur principal : utilisateur

Pré-condition : Ouverture du compte

Post-condition : Ajout d'un nouvel article est établis

Tableau 3-2 : *Scénario de cas d'utilisation ajout article*

Actions acteur	Réactions du système
1. L'utilisateur choisit le service de l'ajout des articles.	2. Le système lui affiche un formulaire lui permettant d'introduire les données convenables.
3. L'utilisateur saisit les informations adéquates.	4. Le système vérifie la violation des contraintes. 5. Le système affiche un message validant l'insertion.
6. L'utilisateur passe à l'article suivant et le processus s'enchaîne au point 2.	

➢ Scénario alternatif :

- Des champs obligatoires non remplis : le scénario alternatif démarre à l'étape 3. Le système indique à l'utilisateur qu'il y a un ou plusieurs champs non remplis. Le scénario nominal démarre à l'étape 3.

- Violation de la contrainte d'unicité d'une clé primaire : le scénario alternatif démarre à l'étape 2. Le système indique à l'utilisateur que l'opération d'insertion a échoué. Le scénario nominal démarre à l'étape 3.

3.2 Diagramme de séquence

Les principales informations contenues dans un diagramme de séquence sont les messages, présentés dans un ordre chronologique, échangés entre l'acteur principal et le système [21].

En effet, nous allons présenter les principaux scénarios de notre application. Les entités qui figurent dans les diagrammes de séquences respectent bien les contraintes imposées par l'architecture J2EE. Ainsi, ces entités appartiennent à plusieurs couches possibles telles que :

- ✓ La couche de présentation contenant les interfaces utilisateurs.
- ✓ La couche métier contenant les composants de contrôle : gestion de flux de contrôle et l'acheminement des requêtes/réponses dans l'application.
    - ✓ Une couche de services : la couche intermédiaire proposée par le Framework Spring qui joue le rôle d'un intermédiaire entre la couche métier et la couche accès aux données.
    - ✓ Une couche DAO : contenant les composants d'accès aux données communiquant avec le SGBD.
- ✓ Les objets entités représentant le résultat de projection du relationnel à l'objet (*Object Relational Mapping*) qui est assurée par Hibernate.

Le scénario correspondant au cas d'utilisation « S'authentifier » montre que l'acteur principal est l'utilisateur. Ce dernier, saisit le login et le mot de passe à partir de l'interface Connexion. Le conteneur d'exécution récupère le contenu du formulaire et invoque la méthode de connexion en lui passant comme paramètres le login et le mot de passe déjà fournis. Le filtre de servlets intercepte les requêtes pour valider les paramètres d'accès à l'application. Nous expliquons ce concept de sécurité dans le chapitre 4.

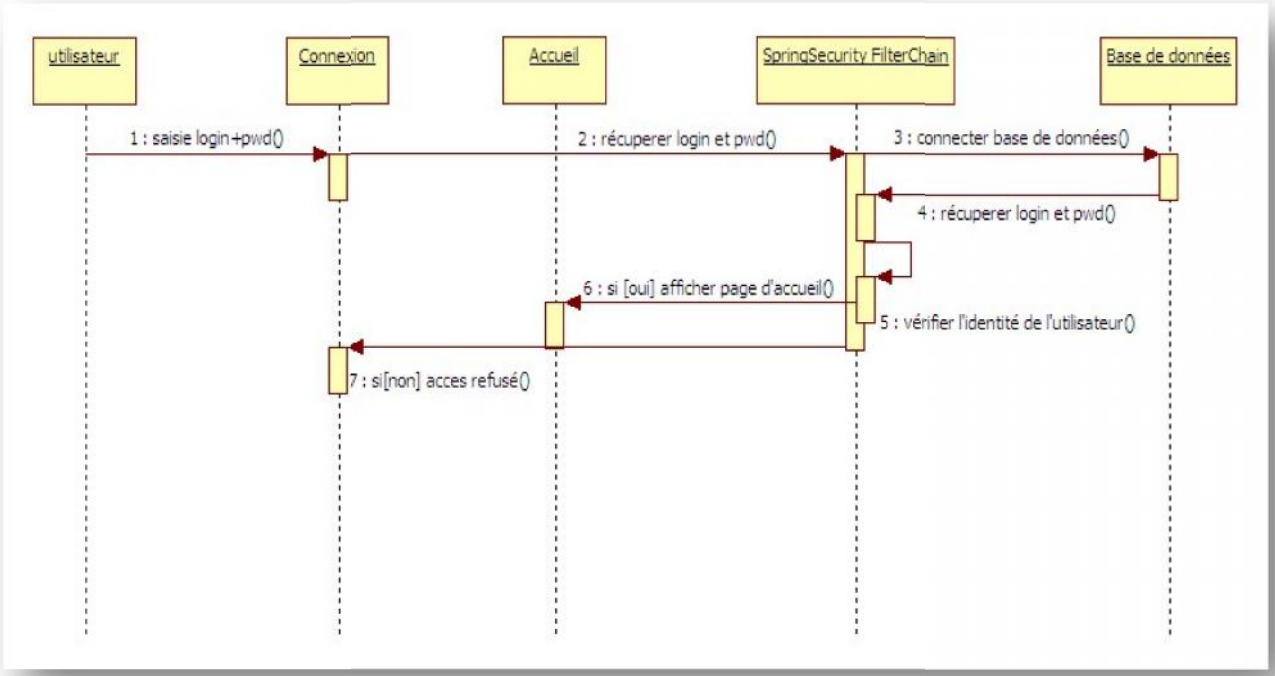


Figure 3-6 : ***diagramme de séquence authentification***

Le diagramme de séquence de la figure 3.7 correspond au cas d'utilisation « Ajouter un article». L'utilisateur saisit le formulaire d'un nouvel article à partir de l'interface AjoutArticle. Suite à l'action de l'ajout, le contrôleur récupère le formulaire et renvoie la requête à la classe ArticleBean qui va invoquer la méthode correspondante de l'interface ArticleService. Dans son implémentation, ArticleService fait appel à l'interface ArticleDao qui va vérifier les contraintes d'intégrité via une méthode adéquate de son implémentation. Ensuite, si les contraintes sont validées, le nouvel article est enregistré et un message de confirmation sera affiché. Si les contraintes sont violées, un message d'erreur est renvoyé.

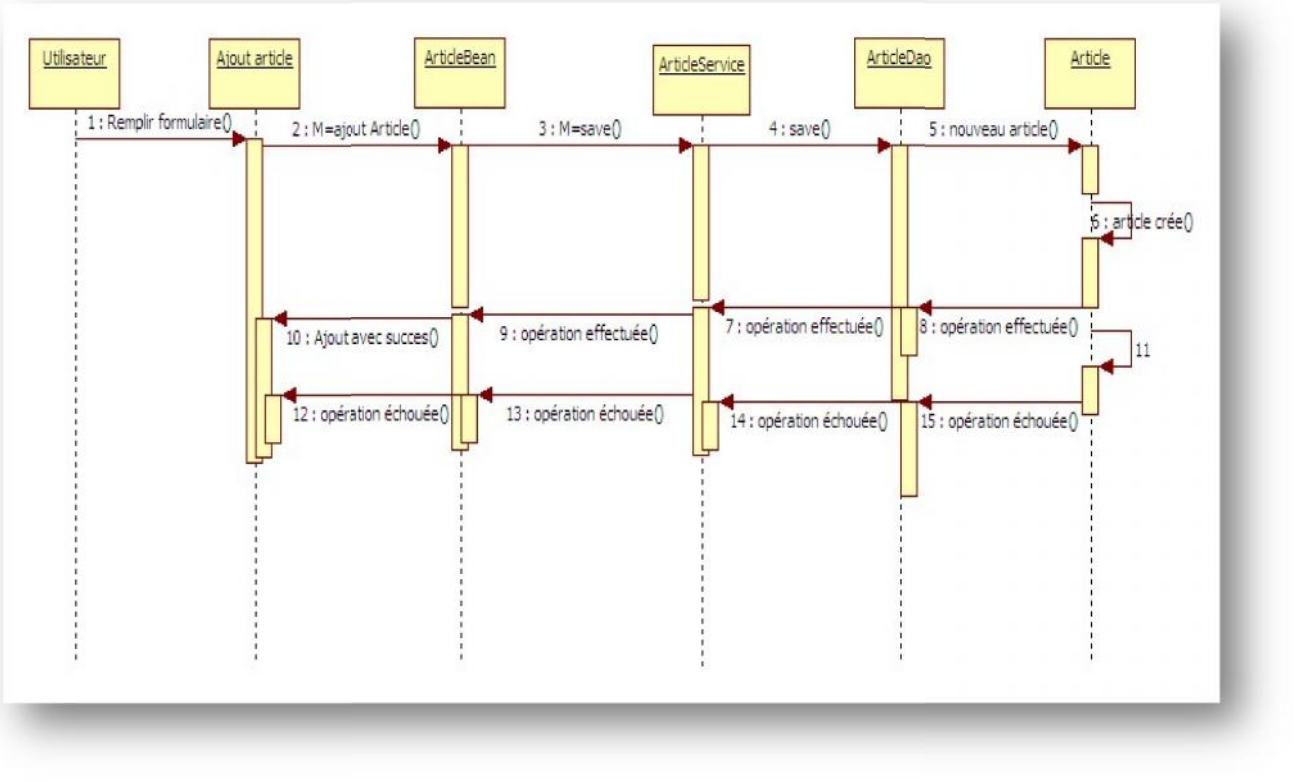


Figure 3-7 : ***Diagramme de séquence ajout article***

3.3 Diagramme de classes

Le diagramme de classes est utilisé pour présenter les classes, les interfaces d'un système ainsi que les différentes relations entre celles-ci. Ce diagramme représente la partie statique d'UML, car il fait abstraction des aspects temporels et dynamiques [21].

Le diagramme de classe de notre application est illustré par la figure 3.8. En effet, ce diagramme résume les aspects fonctionnels de notre application.

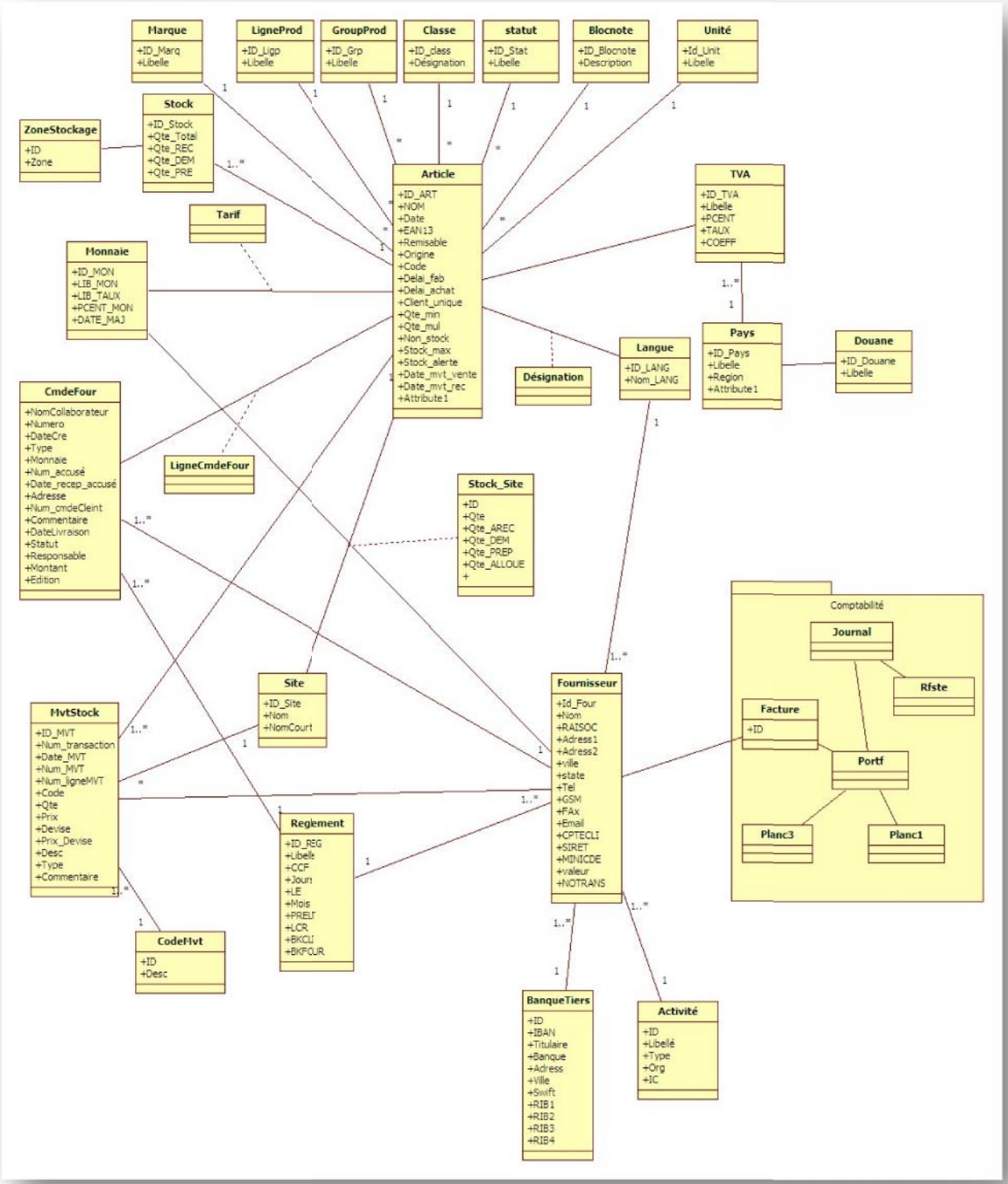


Figure 3-8 : *Diagramme de classes*

4. Conception de la base de données

Avant de mettre en place la base de données, nous avons passé par une étape de modélisation pour faire correspondre le monde réel à une représentation compréhensible par

l’outil informatique. Pour cela, nous avons opté la méthode Merise [22] pour modéliser la base de données de notre système.

4.1 La méthode de Merise

Merise est une méthode de conception, de développement et de réalisation de projets informatiques. Le but de cette méthode est d'arriver à concevoir un système d'information. La méthode Merise est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques [22].

La séparation des données et des traitements assure la structure de notre modèle utilisé (MVC). En effet, l'agencement des données n’est pas souvent modifié, tandis que les traitements le sont le plus fréquemment.

4.2 Le modèle physique des données

Le modèle physique des données est un outil de conception de base de données qui permet de définir la mise en œuvre de structures physiques et de requêtes portant sur les données.

Le modèle physique permet de concevoir la structure d'une base de données destinées à gérer de gros volumes de données opérationnelles. La figure 3-9 illustre le modèle physique des données.

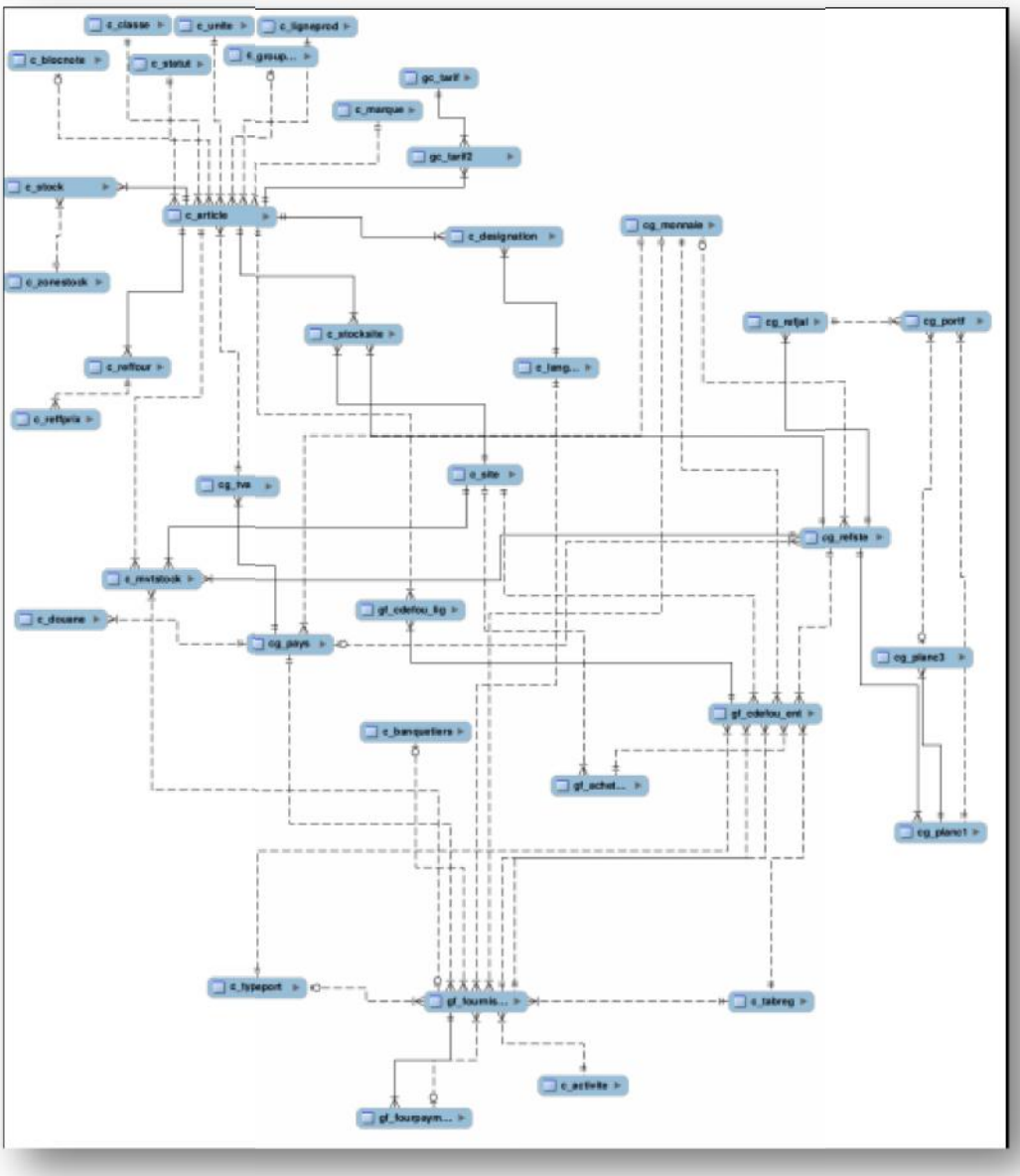


Figure 3-9: *Modèle physique de données*

5. Conclusion

Dans ce chapitre nous avons présenté la spécification détaillée de notre application. Nous avons pu modéliser notre projet tout en élaborant les diagrammes de cas d'utilisation, de séquence et de classes.

Nous avons pu également importer des données en se basant sur le modèle physique des données. Cette modélisation nous a servit dans la réalisation et l'implémentation de l'application qui sera présentée dans le chapitre 4.

Chapitre 4 : Réalisation

Introduction

Une des étapes du cycle de vie d'un projet, aussi importante que la conception, est l'implémentation. Cette étape constitue la phase d'achèvement et d'aboutissement du projet. Pour accomplir cette tâche avec succès, il est nécessaire d'utiliser des outils adéquats.

Ce chapitre présente tout d'abord l'environnement technique du travail ainsi que le choix pris en matière d'environnement logiciel. Par la suite, nous présentons une démonstration pour le test de fonctionnement de notre plateforme.

1. Etude technique

1.1 Environnement de travail

NetBeans est un environnement de développement intégré (EDI). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne [23].

NetBeans supporte une importante variété d'environnements pour l'exécution d'applications web et Java EE: Java Server Pages (JSP), Java Server Faces(JSF), Enterprise JavaBeans (EJB 2.1, EJB 3, EJB 3.1), Apache Struts, Spring Web MVC, Hibernate, etc. Il supporte les standards Java EE 6, Java EE 5, J2EE 1.4, Java Persistence API (JPA 2.0), Java Servlet API.

1.2 Serveur de déploiement

Apache Tomcat [24] est un serveur de déploiement mis en place pour déployer les applications J2EE avec le support de nouvelles fonctionnalités pour abaisser le coût des opérations, l'amélioration des performances et pour soutenir le portefeuille des applications web. Dans notre cas, il représente un conteneur de Servlet J2EE qui implémente la spécification des Servlets et des JSP de Sun Microsystems.

1.3 PostgreSQL

PostgreSQL [25] est un système libre de gestion de base de données relationnelle et objet (SGBDRO). Il fait partie des logiciels de gestion de base de données les plus utilisés dans le monde, autant par le public (applications web principalement) que par des professionnels, en concurrence avec Oracle et Microsoft SQL Server.

1.4 SVN(Subversion)

Subversion est un système de gestion de version. Concrètement, ce système permet aux membres d'une équipe de développeurs de modifier le code du projet quasiment en même temps. En effet, le projet est enregistré sur un serveur SVN et à tout moment. Le développeur peut mettre à jour une classe avant de faire des modifications pour bénéficier de la dernière version. Il peut également comparer deux versions d'un même fichier. Dans notre cas, le travail est réparti sur différents sites locaux. Par conséquent, nous avons utilisé SVN pour la sauvegarde et la restauration des dernières versions sans avoir à copier et coller le projet d'un endroit à un autre. Le serveur SVN étant installé sur un serveur de la société et accessible à partir d'une connexion sécurisée VPN. Le principe de fonctionnement est illustré dans la figure 4.1.

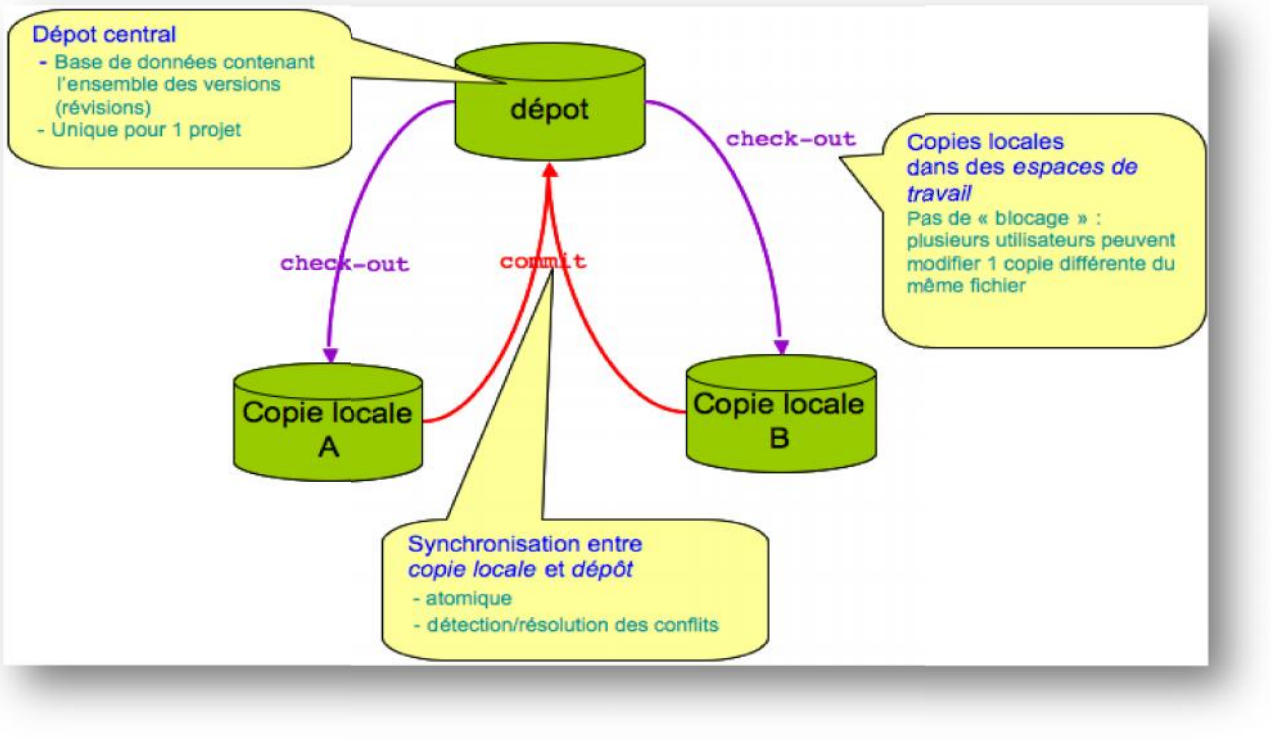


Figure 4-1 : *Fonctionnement du système SVN*

1.5 IReport

IReport est un outil de conception WYSIWYG (*What You See Is What You Get*) exclusivement réservé à la création de fichier de description pour JasperReports [26]. Ainsi, il permet de produire de manière assez intuitive des fichiers .jrxml (fichiers xml) exploitables par JasperReports pour générer des rapports au sein d'une application Java. Le format de rapport généré dépend de JasperReports et du code utilisé (html, pdf, etc.). JasperReports [26]

est un outil de *Reporting Open Source*, offert sous forme d'une bibliothèque qui peut être embarquée dans tous types d'applications Java. Il offre de nombreuses possibilités telles que l'export de rapports aux formats PDF, HTML, XLS, CSV, XML, RTF et TXT.

1.6 Maven

Maven est un outil open-source de *build* pour les projets Java très populaire, conçu pour supprimer les tâches difficiles du processus de *build*. Maven utilise une approche déclarative, où le contenu et la structure du projet sont décrits, plutôt qu'une approche par tâche utilisée par exemple par *Ant* ou les fichiers *make* traditionnels. Cela aide à mettre en place des standards de développement au niveau d'une société et réduit le temps nécessaire pour écrire et maintenir les scripts de *build* [27].

1.6.1 Le modèle objet projet

Le cœur d'un projet Maven 2 est le modèle objet projet (appelé POM, *project object model*). Il contient une description détaillée de notre projet, avec des informations concernant en particulier le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure etc.. Le POM prend la forme d'un fichier XML (*pom.xml*) qui est placé dans le répertoire de base de notre projet.

1.6.2 Le cycle de vie du projet

Les cycles de vie des projets sont un concept central de Maven 2. Maven a pour objectif de gérer tout le cycle de vie de l'application. Pour cela, il définit une liste d'étapes ordonnées constituant le cycle de vie par défaut d'un projet. Chaque type de projets (jar, war, etc.) pourra associer des actions différentes pour chaque étape. La figure 4-2 montre les phases les plus utiles du cycle de vie Maven 2.

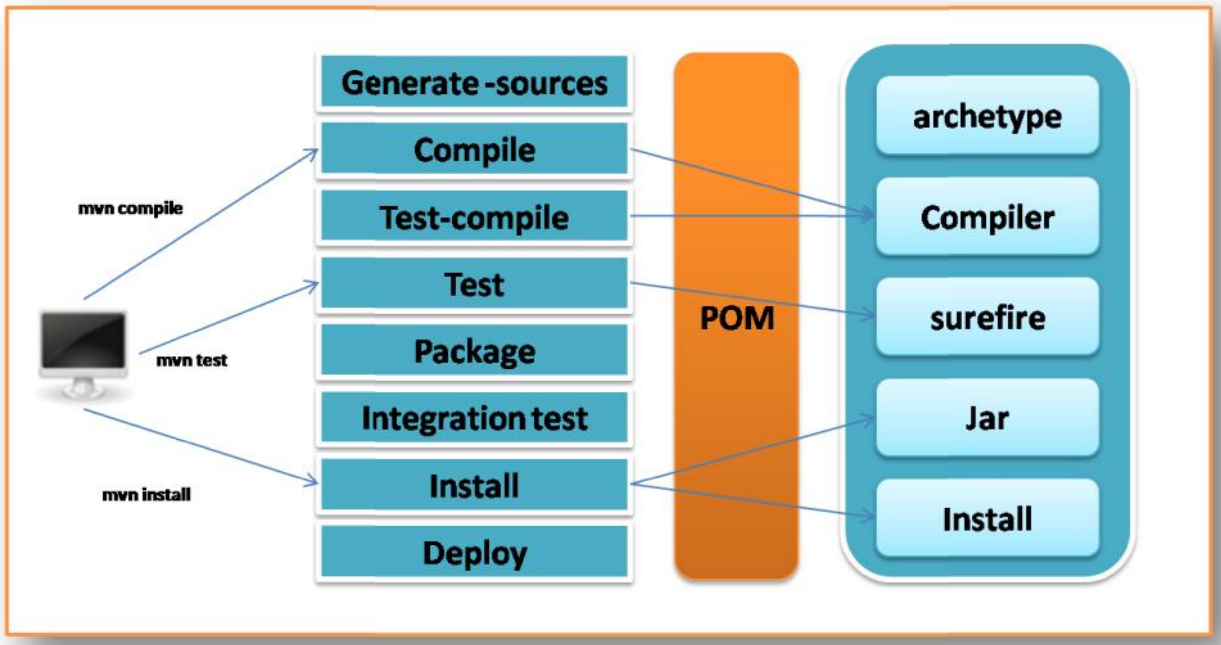


Figure 4-2 *Les phases du cycle de vie maven2*

- *generate-sources*: génère le code source supplémentaire requis par l'application, ce qui est généralement accompli par les plug-ins appropriés.

- *compile*: compile le code source du projet.

- *test-compile*: compile les tests unitaires du projet.

- *test*: exécute les tests unitaires (typiquement avec Junit) dans le répertoire src/test.

- *package*: met en forme le code compilé dans son format de diffusion (JAR, WAR, etc.).

- *integration-test*: réalise et déploie le package si nécessaire dans un environnement dans lequel les tests d'intégration peuvent être effectués.

- *install*: installe les produits dans l'entrepôt local, pour les utiliser comme dépendance des autres projets sur la même machine.

- *deploy*: réalisé dans un environnement d'intégration ou de production, il copie le produit final dans un entrepôt distant pour être partagé avec d'autres développeurs ou projets.

2. Production des programmes

2.1 Architecture 3-tiers et mise en place du modèle MVC

Notre application web est bâtie sur une architecture 3-tiers en respectant le modèle MVC et la façon de la mise en place des *Framework* Spring, Hibernate et Richfaces.

Pour ce faire, notre application est constituée de deux grands modules (figure 4-4).

Le premier modules étant *DiagErp Core*, incluant trois packages :

- ✓ Entity : ce répertoire regroupe les entités et les fichiers de correspondance O/R (Objet/Relationnel)
- ✓ DAO : dans ce package existe toutes les interfaces et leurs implémentations dont le rôle est d'assurer l'accès aux données de la base de données (Data Access Object).
- ✓ Service : Conformément à la structure du Framework Spring, ce package est constitué par des interfaces associées à leurs implémentation afin d'apporter l'aspect de l'inversion de contrôle de Spring (IoC). Dans chaque implémentation, nous retrouvons une référence de l'interface correspondante dans le package DAO pour maintenir l'enchaînement lors de l'accès à la base de données.

Le deuxième module étant *DiagErp Web* dans lequel nous retrouvons les différents packages. Le deuxième est le package des Beans incluant l'ensemble des classes java qui représentent la couche métier. Le deuxième package contient l'ensemble des interfaces graphiques qui constituent notre application web.

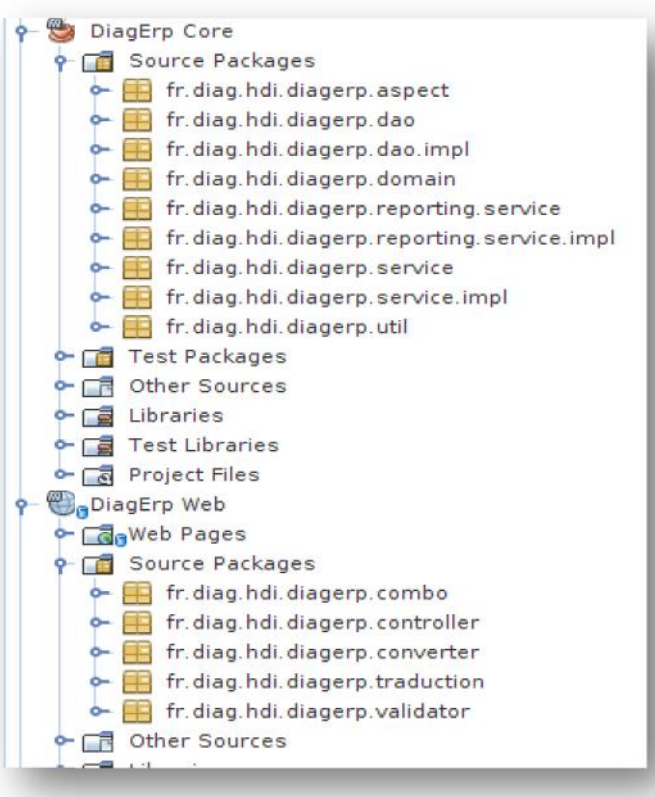


Figure 4-3 ***Structure générale de l'application***

La figure 4-4 montre l'architecture 3-tiers de la plateforme J2EE sur laquelle repose notre application Web sur la gestion commerciale. En effet, elle se compose de trois couches: présentation, métier, accès aux données et persistance. L'utilisation d'une telle architecture permet de bien séparer les rôles de chaque membre dans une équipe de développement.

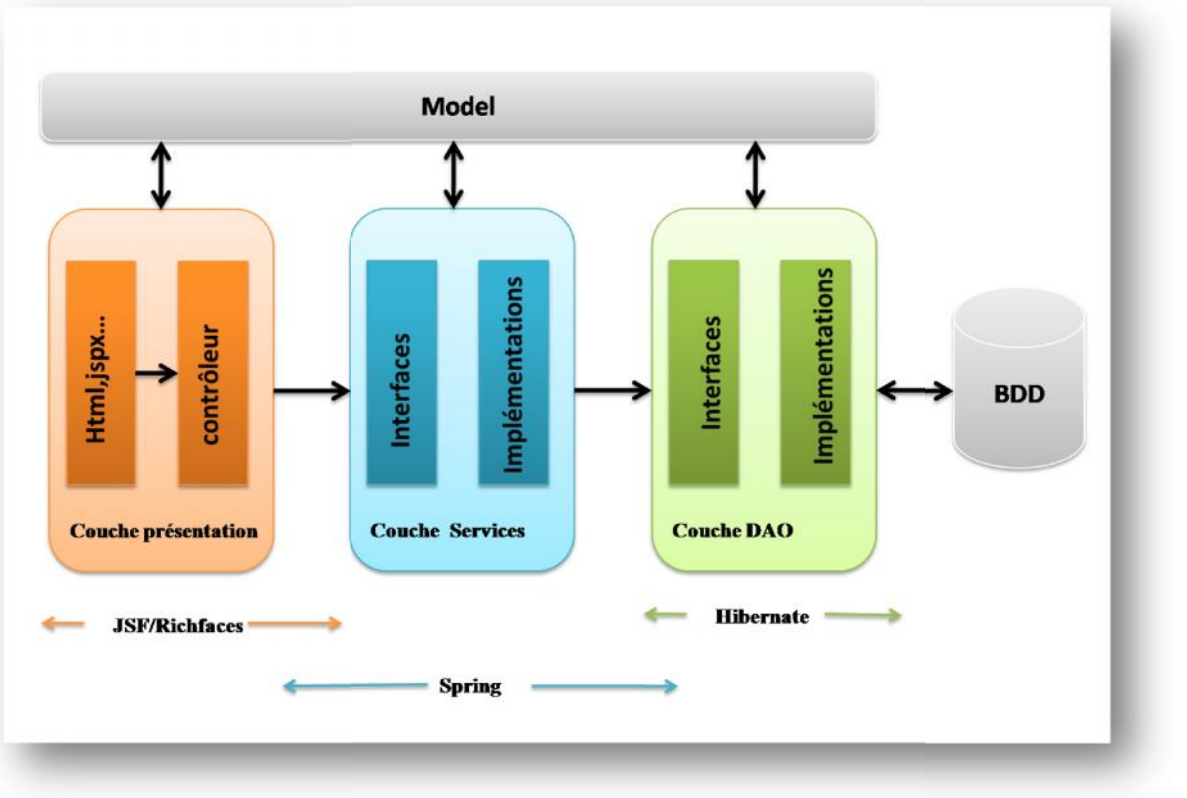


Figure 4-4 : *architecture générale de l'application*

En conclusion, nous avons constaté qu'il y a quatre couches principales dans notre architecture déployée:

- ✓ La couche DAO qui s'occupe de l'accès aux données, le plus souvent des données persistantes au sein d'un SGBD.
- ✓ La couche Service qui est la couche intermédiaire entre la couche métier et la couche DAO. La couche Service assure l'inversion de contrôle (IoC).
- ✓ La couche métier (Model) implémente les algorithmes « métiers » de l'application. Cette couche est indépendante de toute forme d'interface avec l'utilisateur. Ainsi elle doit être utilisable avec une interface web. Elle devrait être testée en-dehors de l'interface web et notamment avec une interface console. C'est généralement la couche la plus stable de l'architecture. Elle ne change pas en cas de changement de l'interface utilisateur ou même à la façon d'accéder aux données nécessaires au fonctionnement de l'application.
- ✓ La couche présentation qui est une page xhtml permet à l'utilisateur de piloter l'application et d'en recevoir des informations.

Les couches métier et DAO sont utilisées via des interfaces Java. Ainsi, la couche métier ne référence pas la couche DAO. À ce niveau, les interfaces sont uniquement manipulées sans passer par les implémentations. Ceci assure l'indépendance des couches entre-elles : changer l'implémentation de la couche DAO n'a aucune incidence sur la couche métier tant qu'on ne touche pas à la définition de l'interface de la couche DAO. Il en est de même entre les couches interface utilisateur et métier.

2.2 Spring

Spring est une plate-forme qui se base sur la notion de conteneur léger par opposition au conteneur EJB [16] considéré lourd technologiquement et surtout peu adaptif par rapport aux différents types de problématiques courantes.

2.2.1 La notion de conteneur léger

Comme nous l'avons mentionné à la section 2.2, Spring repose sur un conteneur léger. Ce dernier, fournit un support simple et puissant pour gérer une application via un ensemble de composants, c'est-à-dire des objets présentant une interface dont le contenu interne n'est pas connu par les autres composants. Le conteneur léger gère le cycle de vie des composants (création, destruction) mais aussi leurs interdépendances (un composant devrait s'appuyer sur d'autres pour fonctionner).

Le conteneur léger permet d'avoir des applications plus portables, c'est-à-dire parfaitement indépendantes du serveur d'applications. En effet, l'application vient avec son propre conteneur qui lui fournit l'infrastructure dont elle a besoin.

À travers son approche par composants, le conteneur léger encourage les bonnes pratiques de programmation : par interface et à faible couplage. Cela assure une meilleure évolutivité des applications mais aussi améliore grandement leur testabilité.

2.2.2 L'écosystème Spring

Comme le montre la figure 4-5, le projet Spring est constitué des modules suivants :

- Core, le noyau, qui contient à la fois un ensemble de classes utilisées par toutes les briques du *framework* et le conteneur léger.
- AOP : le module de programmation orientée aspect, qui s'intègre fortement avec AspectJ, un *framework* de POA à part entière.
- DAO (Data Access Object): qui constitue le socle de l'accès aux dépôts des données, avec notamment une implémentation pour JDBC. D'autres modules fournissent des

abstractions pour l'accès aux données (solutions de mapping Objet-relationnel, LDAP) qui suivent les mêmes principes que le support JDBC. La solution de gestion des transactions de Spring fait aussi partie de ce module.

- ORM (Object Relationnal Mapping) : qui propose une intégration avec des outils populaires de mapping objet-relationnel, tel que Hibernate, JPA (Java persistance api), EclipseLink ou iBatis. Chaque outil peut bénéficier de la gestion des transactions fournie par le module DAO.
- Java EE, un module d'intégration d'un ensemble de solutions populaires dans le monde de l'entreprise.
- Web, le module d'intégration d'un ensemble de solutions populaires dans le monde de l'entreprise.
- Web, le module comprenant le support de Spring pour les applications Web. Il contient notamment Spring Web MVC, la solution de Spring pour les applications Web, et propose une intégration avec de nombreux *frameworks* Web et des technologies de vue.
- Test, qui permet d'appliquer certaines techniques de conteneur léger Spring aux tests unitaires via une intégration avec JUnit et TestNG.

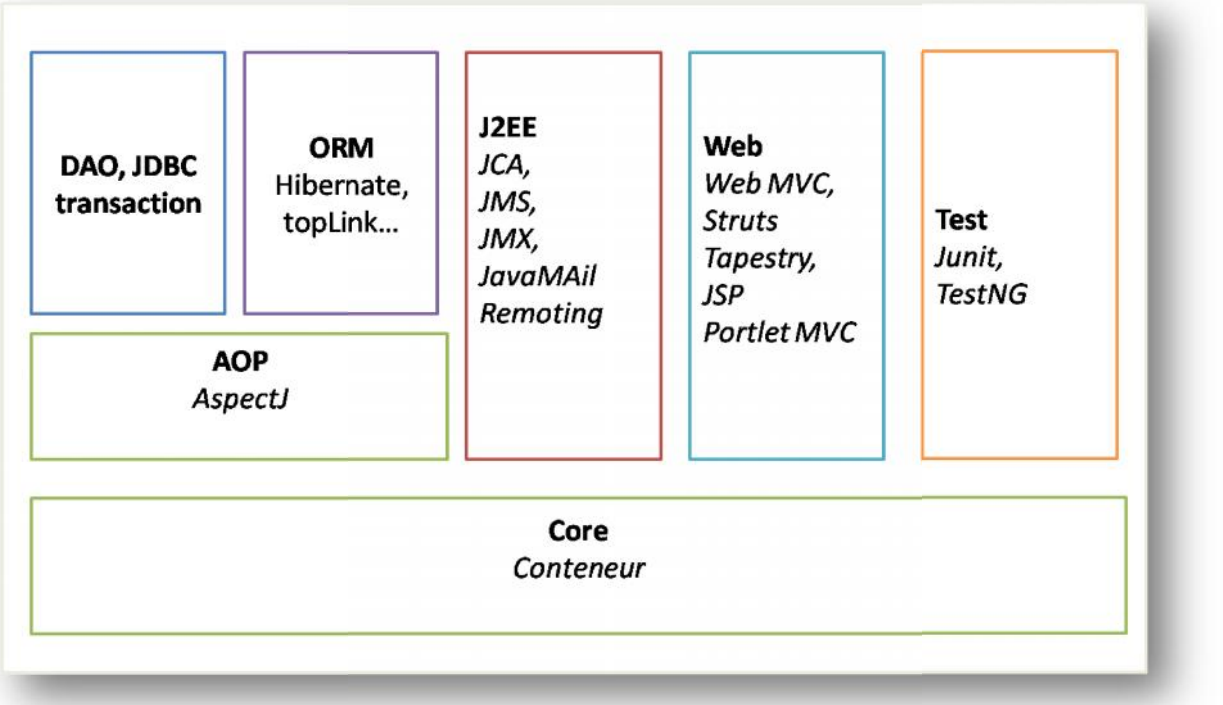


Figure 4-5 : *core spring*

2.2.3 Concepts clés Spring

2.2.3.1 L'inversion de contrôle

Les conteneurs légers sont souvent appelés conteneurs d'inversion de contrôle, ou IOC (Inversion of Control). Dans cette section, nous présentons l'origine de ce concept et dans quelle mesure il se rapproche de la notion de conteneur léger.

2.2.3.2 L'injection de dépendances

Dans son principe, l'injection de dépendance s'appuie sur un objet assembleur, le conteneur léger, capable de gérer le cycle de vie des composants d'une application, ainsi que leurs dépendances, en les injectant de manière appropriée.

Il existe différents moyens d'effectuer l'injection de dépendances. Spring utilise l'injection par constructeur (un objet se voit injecter ses dépendances au moment où il est créé, c'est-à-dire via les arguments de son constructeur) et l'injection par modificateurs (un objet est créé, puis ses dépendances lui sont injectées par les modificateurs correspondants). Les deux formes ne sont pas exclusives, et il est possible de les combiner. Grâce à l'injection de dépendances, les composants sont plus indépendants de leur environnement d'exécution. Ils n'ont pas à se soucier de l'instanciation de leurs dépendances et peuvent se concentrer sur leur tâche principale. De plus, l'injection de dépendances mettant en jeu le conteneur léger, c'est lui qui gère toutes les problématiques de configuration, facilitant, par exemple l'externalisation des paramètres.

L'injection de dépendance met en jeu un référentiel de description des dépendances. Avec Spring, sa forme la plus connue est XML, mais il est possible d'utiliser d'autres formes, comme de simples fichiers texte ou même du Java. L'essentiel est de disposer au final d'un objet contenant tout les composants d'une application correctement initialisés.

2.2.4 Configuration de Spring

La configuration de Spring passe par un fichier de configuration XML (Application Context.xml). Il faudra éventuellement faire les déclarations nécessaires. En premier lieu, nous allons aborder la manipulation au niveau de web.xml. En second lieu, nous allons traiter le fichier de configuration de Spring.

Web.xml

Les balises suivantes doivent êtres inscrites dans le descripteur de déploiement :

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
</servlet>
```

Figure 4-6 *Déclaration de fichier spring dans le fichier web.xml*

Pour le fichier Spring, une première étape consiste à définir les paramètres de connexion à la base de données comme montré dans la figure 4.7.

```
<!-- DATA SOURCE CONFIGURATION -->
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
        <value>${jdbc.driverClassName}</value>
    </property>
    <property name="url">
        <value>${jdbc.url}</value>
    </property>
    <property name="username">
        <value>${jdbc.username}</value>
    </property>
    <property name="password">
        <value>${jdbc.password}</value>
    </property>
</bean>
```

Figure 4-7 : ***configuration des paramètres de connexion à la base de données***

Ce fichier décrit les objets à fabriquer et les relations de dépendances entre eux. L'idée du modèle IoC est très simple, elle consiste à déléguer à un objet C la mise en relation de A avec B, lorsqu'un objet A a besoin d'un objet B.

En outre, notre fichier de configuration contient la configuration nécessaire pour implémenter Hibernate (source de données, paramètres de connexion, et la définition des fichiers de projection).

```
  <!--Hibernate Template Defintion-->
<bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
  <property name="sessionFactory">
    <ref bean="sessionFactory"/>
  </property>
  <property name="jdbcExceptionTranslator">
    <ref bean="jdbcExceptionTranslator"/>
  </property>
</bean>
```

Figure 4-8: ***configuration Hibernate***

2.3 Spring Security

Les premiers services de sécurité introduits dans la galaxie Spring étaient supportés par les modules Acegi Security. Ces modules présentaient une relative complexité pour leur mise en œuvre et restaient spécifiques. La montée en puissance des besoins en termes de sécurité des applications et l'enrichissement du catalogue des solutions ont conduit à une standardisation de ces modules qui sont devenus Spring Security.

2.3.1 Fonctionnalités

Les fonctionnalités adressées par les composants de Spring Security concernent uniquement deux volets des services de sécurité : identification et authentification des utilisateurs d'une part et habilitations pour l'accès ou non aux différents services applicatifs, d'autre part.

2.3.1.1 Identification et authentification

Les différents modes classiques d'identification et d'authentification des utilisateurs sont supportés par Spring Security : basic, form et digest.

De même plusieurs modèles de référentiels utilisateurs sont accessibles à travers les composants de Spring Security (une liste donnée directement en configuration à des fins de tests, une base de données, un annuaire LDAP, etc.). La configuration des composants Spring Security par injection de dépendances permet une grande souplesse de changements entre ces possibilités.

2.3.1.2 Habilitations

Pour la gestion des habilitations, Spring Security supporte notamment les mécanismes liés aux rôles des utilisateurs.

Pour une gestion nettement plus fine des autorisations d'accès, des mécanismes de ACL (Access Control List) sont disponibles non seulement pour la richesse des possibilités offertes par ces mécanismes mais également en raison de la complexité de leur mise en œuvre.

2.3.1.3 Définition des règles dans la sécurité

Les règles dans la sécurité permettent de qualifier un utilisateur par rapport à un rôle ou une ressource par rapport à certaines caractéristiques. Pour définir ces règles, deux modes existent :

- ✓ Le mode déclaratif a priori : l'utilisateur A possède les rôles R1 et R2. Il s'agit là d'une information définie a priori et figée.
- ✓ Le mode reposant sur des expressions évaluées en temps réel : l'utilisateur est authentifié et par conséquent, il peut accéder à la ressource demandée (la qualité anonyme ou authentifié de l'utilisateur est vérifiée au moment voulu).

Spring Security supporte ces deux modes de définition des règles de sécurité. Le premier des deux modes est utilisé préférentiellement dans notre projet.

2.3.2 Principes

Les deux principes de fonctionnement de Spring Security sont les suivants :

- ✓ Les requêtes des utilisateurs sont interceptées lors de leur réception par le conteneur d'exécution. Des masques de sélection permettent de sélectionner les requêtes pour lesquelles des contraintes de sécurité sont définies. Ces dernières sont prises en charge par des composants hébergés dans le conteneur léger pour les traitements de sécurité, les autres sont autorisées par défaut.
- ✓ Pour la gestion des habilitations, les composants de la couche de sécurité votent pour autoriser ou non un accès. Les votes sont collectés par des gestionnaires d'accès qui les consolident selon des politiques telles que l'absence d'interdiction ou la nécessité de l'unanimité.

2.3.3 Installation de la distribution

Les modules Spring Security ne sont pas intégrés dans la distribution principale Spring Framework. La figure 4-9 montre la mise en place du module Spring Security à travers le fichier applicationContextSecurity.xml

```
<security:authentication-manager >
    <security:authentication-provider>
      <security:password-encoder hash="md5"></security:password-encoder>
        <security:jdbc-user-service data-source-ref="dataSource"
                users-by-username-query="SELECT login, password, enabled
                                         FROM utilisateurs WHERE utilisateurs.login = ?"
                authorities-by-username-query="SELECT utilisateurs.login, role
                                               FROM utilisateurs,roles WHERE utilisateurs.login=roles.login AND utilisateurs.login = ?"
                 />

    </security:authentication-provider>

</security:authentication-manager>
```

Figure 4-9 *Configuration de Spring Security*

2.4 Hibernate

Lors de l'utilisation du Framework, nous étions face à un paramétrage nécessaire :

- ✓ La définition du modèle de classes exploitant la base de données.
- ✓ Une mise en correspondance entre le modèle de classes et la base de données.

En outre, après la phase de paramétrage, plusieurs fonctionnalités étaient présentes pour faciliter notre phase d'implémentation. Ces fonctionnalités sont les suivantes :

- ✓ Rendre persistant un objet d'une classe : c'est la méthode *save* qui nous a offert cette fonctionnalité. Elle prend en paramètre l'objet à rendre persistant.
- ✓ Charger un objet d'une classe à partir de la base de données : nous avons utilisé la méthode *load* à cette finalité.
- ✓ Mettre à jour un objet persistant : il suffit de modifier la valeur des propriétés d'un objet puis d'appeler la méthode *update* de l'objet.
- ✓ Supprimer un objet persistant : l'appel de la méthode *delete* avec en paramètre un objet persistant se charge d'effectuer la suppression dans la base de données.
- ✓ Rechercher des objets : Hibernate propose un langage de requête orienté objets nommé HQL (Hibernate Query Language) dont la syntaxe est similaire au SQL et qui permet d'effectuer des requêtes sur le modèle objet.

Ainsi, nous indiquons que ces méthodes permettent de manipuler les données projetées sur la base de données d'une façon transparente à l'utilisateur. Cette projection est totalement prise en charge par Hibernate.

2.5 JSF, Richfaces et la mise en place du modèle MVC2

Richfaces et JSF sont deux Frameworks utilisés pour développer des applications web J2EE. Ils utilisent et étendent l'API Servlet afin d'encourager les développeurs à adopter l'architecture MVC2. En effet, pour simplifier la réalisation d'un tel modèle, nous avons introduit cette nouvelle version. C'est exactement le même modèle de conception que MVC, à la différence qu'il n'y a plus qu'un seul contrôleur qui se charge de rediriger la requête vers le bon traitement. Pour ce faire, nous obtiendrons l'architecture MVC2 avec Richfaces et JSF via l'utilisation d'un seul fichier de configuration qui est bâtit sur une structure XML (faces-config.xml). Dans ce fichier, nous devons déclarer les classes métier (beans) et les navigations entre les pages (figure 4-10).

```
<managed-bean>
    <managed-bean-name>carticleBean</managed-bean-name>
    <managed-bean-class>fr.diag.hdi.diagerp.controller.CArticleBean</managed-bean-class
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
        <property-name>carticleService</property-name>
        <value>#{carticleService}</value>
    </managed-property>
</managed-bean>
<managed-bean>
    <managed-bean-name>cgTvaBean</managed-bean-name>
    <managed-bean-class>fr.diag.hdi.diagerp.controller.CgTvaBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
```

Figure 4-10 : *configuration des managed Bean*

En plus, nous étions face à une automatisation de la gestion de certains aspects tels que la validation ou la conversion des données entrées par les utilisateurs via l'interface de l'application. Ainsi, nous n'avons plus besoin de coder le contrôle de chaque donnée fournie par un utilisateur, il suffit de décrire les vérifications à effectuer dans le fichier de configuration dédié à cette tâche. Chaque interface utilisateur lui est associée un ou plusieurs *bean*. Pour ce faire, après sa récupération, le contrôleur se charge de rediriger la requête vers le bon traitement.

Ainsi, dans cet exemple de la figure 4.10, pour l'interface utilisateur (article.xhtml) est associé le bean articleBean. Ce dernier contenant des propriétés telles que (articleService) faisant référence à l'interface ArticleService.java). L'interface (ArticleService.java) fait référence elle-même à l'interface DAO qui assure la persistance des objets.

2.5.1 Facelets

Facelets est un framework permettant de créer des vues JSF basées sur le modèle html. Il permet de coder des vues avec de simples balises XML, bien plus familières que le codage de JSP [27].

Le support de XML permet de créer des pages XHTML propres. Il offre également un support à JSF pour l'utilisation de Template ainsi que la possibilité de créer ses propres composants dans une librairie. Ce qui réduit la quantité de code produit et permet la réutilisation du code. Bien que les premières pages qui utilisent JSF étaient des pages JSP, la tendance est de tourner en faveur des facelets, car ces dernières, respectent le cycle de vie JSF et ont été spécialement développé pour être utilisés avec JSF. Ce qui n'est pas le cas pour une page JSP.

A l'origine JSP était utilisé pour simplifier l'utilisation des servlets, car finalement chaque page JSP va être compilée en une servlet.

2.6 Description du processus de développement

Pour créer notre application web en implémentant les Frameworks JSF, Richfaces, Hibernate et Spring, nous avons adopté le processus de développement suivant :

- Génération des fichiers de correspondance
- Intégration de Spring
- Création de la couche DAO et la couche service
- Création de la couche métier
- Création des interfaces graphiques
- Ajout des navigations entre les pages
- Déploiement de l'application

2.6.1 Génération des fichiers de correspondance

Après avoir crée la base de données de notre application, nous devons générer les fichiers de correspondance et les classes entité correspondantes. Deux fichiers en résultent :

- ✓ Un fichier ayant l'extension (.hbm.xml).
- ✓ Un fichier (.java) c'est l'entité qui lui correspond.

2.6.2 Intégration de Spring

Une première étape nécessite l'intégration de Spring en ajoutant sa librairie. Une seconde étape consiste à la création d'un fichier de configuration « applicationContext.xml ».

2.6.3 Création de la couche DAO et la couche service

Cette étape est primordiale, elle permet d'assurer la notion d'inversion de contrôle (IoC). Pour ce faire, nous avons créé des interfaces contenant les signatures des méthodes d'accès et pour chaque interface nous implémentons le code adéquat.

2.6.4 Création de la couche métier

La couche métier contient tout le traitement nécessaire. Cette couche ne dialogue pas directement avec la couche DAO, mais via la couche service qui agit en tant qu'intermédiaire pour les interconnecter.

2.6.5 Création des interfaces graphiques

Nous avons utilisé les Frameworks JSF et Richfaces pour concevoir les interfaces graphiques de notre application.

2.6.6 Ajout des navigations entre les pages

Dans notre application, les règles de navigation (navigation rules) permettent de définir les redirections entre les pages web.

2.6.7 Déploiement de l'application

Le déploiement d'une application web sur un serveur d'applications signifie en quelque sorte son installation. En effet, notre application est déployée sur le serveur Tomcat. À cet effet, nous avons opté à l'utilisation d'un fichier de déploiement dont l'extension est ear. L'idée de l'utilisation d'un archive d'entreprise (ear) était issue du fait que notre application est constituée par deux archives :

- ✓ Jar : représente l'empaquetage du module Model
- ✓ War : représente l'empaquetage du module ViewController qui représente la couche présentation.

3. Implémentation de l'application

Dans ce qui suit, nous allons illustrer quelques écrans qui constituent notre application.

3.1 Interface d'authentification

Une fois le site chargé, le planificateur se trouve face à la page d'accueil lui permettant d'accéder aux différents utilitaires ou services qui lui sont offert.

Figure 4-11 *Interface d'authentification*

3.2 Interface principale

Une fois l'utilisateur est connecté, l'interface principale s'affiche. En effet, chaque interface utilisateur est répartie en trois parties respectant ainsi l'implémentation de notre page Template. Cette dernière, élimine la redéfinition de chaque page. Ainsi, il suffit de faire appel à cette page Template qui contient une entête, notre menu général et une section au centre qui contiendra le contenu que nous voulons afficher.

Le bouton Déconnexion de menu en haut permet à l'utilisateur de se connecter à la session. Ainsi, le menu à gauche permet de naviguer entre les différentes pages de l'application.

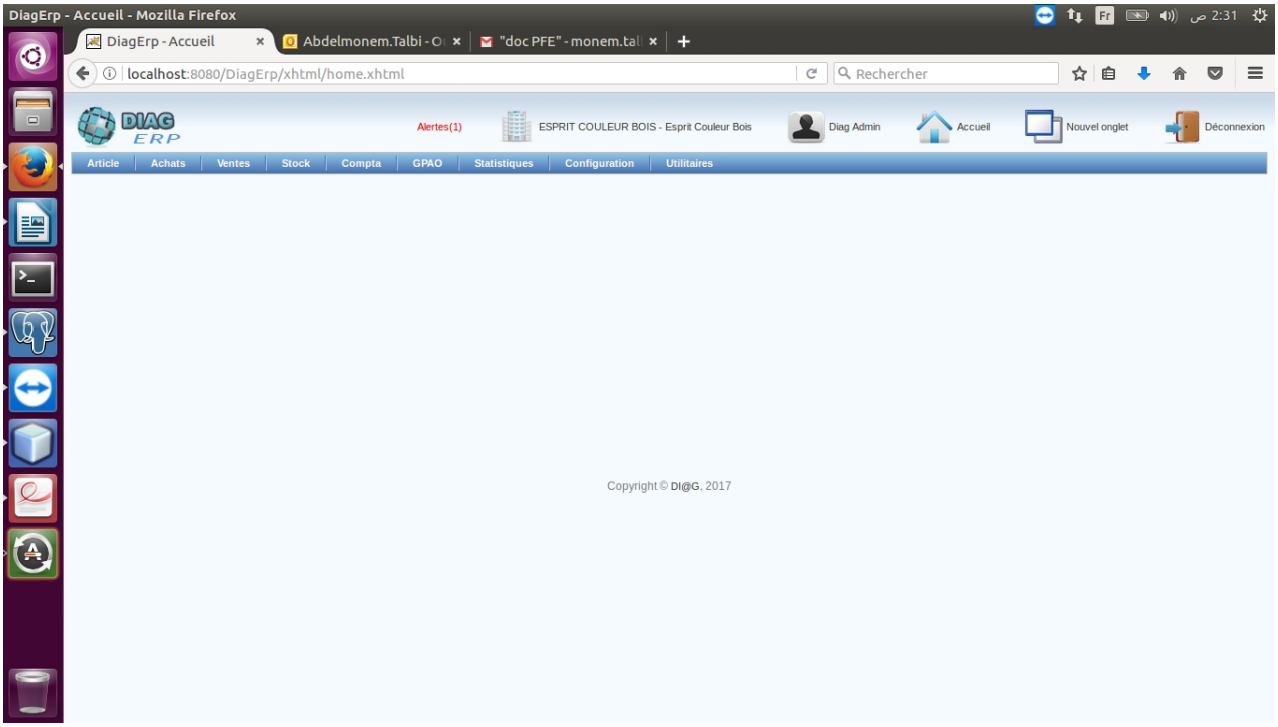


Figure 4-12 *Interface d'accueil*

3.3 Interface article

En cliquant sur la rubrique Article dans le menu principal (à gauche), l'application affiche la liste des articles déjà créés dans une table, l'utilisateur a le choix d'ajouter un nouvel article ou de mettre à jour un article de la liste ou encore de le supprimer. Le panneau de recherche qui existe dans la table permet de filtrer la liste des articles affichés.

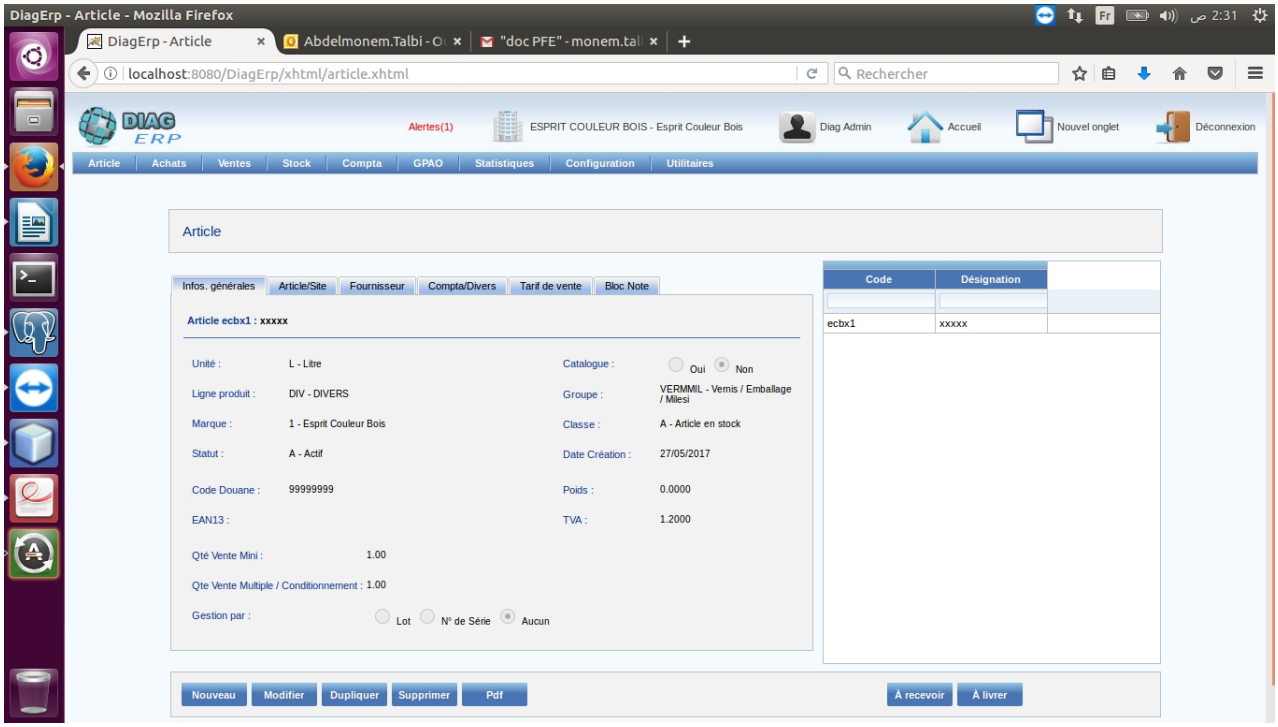


Figure 4-13 *Interface Article*

4. Conclusion

Dans ce chapitre nous avons présenté la phase de réalisation de la solution déjà conçue dans le chapitre 3. Ensuite, nous avons décrit les caractéristiques logicielles de notre environnement, en expliquant en détail la mise en œuvre des technologies utilisées. Par la suite, nous avons donné un aperçu sur les fonctionnalités de l'application en représentant des interfaces représentatives.

Conclusion générale et perspectives

La technologie Web est une technologie assez récente, qui a trouvé des applications spécifiques dans l'entreprise avec l'apparition du concept de l'Extranet. Son adaptation avec les technologies antérieures, particulièrement les technologies client/serveur basées sur les serveurs de données, s'avère prometteuse.

Le présent travail s'insère dans cette thématique. Nous avons pu définir et réaliser une application Web pour la gestion commerciale des entreprises. Elle représente une solution permettant de mieux gérer l'ensemble des processus d'une entreprise intégrant l'ensemble de ses fonctions comme l'approvisionnement et la gestion financière et comptable.

Ce projet était une occasion pour appliquer nos connaissances théoriques acquises et pour tirer profit du travail en groupe ainsi que pour nous initier à réaliser des projets de développement professionnels.

Durant la réalisation de ce projet nous avons essayé de suivre une démarche rationnelle qui se base sur le langage d'analyse et de conception (UML et Merise).

Quant à la phase d'implémentation, nous avons choisi l'architecture J2EE pour l'ergonomie et l'aisance de manipulation qu'elle offre.

L'application réalisée, était tout au long du cycle de développement une réponse au cahier des charges. Aussi, pouvons-nous dire que notre projet était une excellente opportunité pour s'initier à une gestion de projets de grande échelle ? Outre la pratique de programmation, notre satisfaction vient du fait que l'idée principale du projet a été amplement implémentée. Mais, cela n'empêche pas à penser à des améliorations que nous pouvons ajouter afin de répondre mieux aux besoins de différents utilisateurs de notre application. Nous envisageons ainsi d'améliorer notre application en ajoutant d'autres modules comme la gestion de ventes et la gestion des relations clientèles (CRM). Entre autres, notre objectif à court terme est la création d'un module pour la gestion des achats, allant de la gestion du stock jusqu'au module financier pour le calcul du coût d'une telle transaction. Ainsi, nous envisageons que l'internationalisation de notre application, fût une phase primordiale dans l'avenir.

Bibliographie

[1] http://fr.wikipedia.org/wiki/Progiciel_de_gestion_int%C3%A9gr%C3%A9

[2] http://fablain.developpez.com/tutoriel/presenterp/

[3] Caillaud, 04 J.Caillaud « progiciel du système intégré? » définition qui concerne le concept ERP

[ref] : ISO/IEC 14764 - Software Engineering -- Software Life Cycle Processes -- Maintenance

[4] [ERP : GOC (Government Owned Corporation)] précise que les L'ERP sont des progiciels

[5] SUPINFO, Matthieu CIRELLI, « Un ERP, ou Enterprise Resource Planning, est un progiciel de gestion intégré

[6] http://www.acheteursinfo.com/actualites_erp.html

[7] http://www.entreprise-erp.com/articles/marche-des-erp.html

[8] source lesjeudis.com

[9] https://www.lesjeudis.com/article/cb-468-les-principaux-editeurs-derp

[10] http://www.entreprise-erp.com/articles/definition-erp.html

[11] http://www.guideinformatique.com/fiche-erp-297.htm

[12] Livre Blanc Erp OpenSource

[13] http://fablain.developpez.com/tutoriel/presenterp/

[14] http://jmdoudoux.developpez.com/cours/developpons/java/chap-j2ee-javaee.php

[15] http://www.roseindia.net/ejb/introduction/j2eeintroduction.shtml

[16] http://ego.developpez.com/spring/

[17] http://loic-frering.developpez.com/tutoriels/java/hibernate-jpa-spring-tapestry/

[18] ftp://ftp-developpez.com/schmitt/tutoriel/java/jsf/introduction/introduction.pdf

[19] http://baptiste-wicht.developpez.com/tutoriels/conception/mvc/

[20] http://lmellouk.developpez.com/tutoriels/jsf/richfaces/

[21] [GAR, 06] : GARGOURI Faeïz, « Modélisation avec le langage UML : Conception orienté objet des systèmes d'information », cours d'ingénieur, 2007-2008.

[22] https://www.coursehero.com/file/23328601/1-MERISE-Support-M2pdf/

[23] https://fr.wikipedia.org/wiki/NetBeans

[24] https://fr.wikipedia.org/wiki/Apache_Tomcat

[25] https://fr.wikipedia.org/wiki/PostgreSQL

[26] http://jpg.developpez.com/bi/tutoriels/jasperreports/initiation/

[27] http://djo-mos.developpez.com/tutoriels/java/jsf/facelets-intro/