Cahier conception de la solution Billing-system

08/04/2024

Version n°: 1.0

Références du document

Nom	**Cahier d'analyse de la Solution Billing-System**
Auteur	1. Djama Vincent Joël 2. Elvise Ojong Oben **V1.0**

Relecture du document

Nom	Fonction	Date
M. FROUMSIA DOKROM	Project Manager	**********
M. TABE BESSEM	IT Manager	**********
M. BIKADAI RADIUS	Scrum master	**********
M. KOLIAN DAGAULE	Product owner	**********
M. TABI AGWO	Développeur	**********
M. PENN ERIC TANDAH	Développeur	**********
M. PAYANG ARISTIDE	Développeur	**********

Validation du document

Nom	Fonction	Date

Versions

Version	Date	Auteur	Fonction	Commentaire
1.0	08/04/2024	1) Djama Vincent Joël 2) Elvise Ojong Oben	Développeur	Création

Table des Matières

I. DOMAINE D ' APPLICATION....3
1. Objectifs....3
2. Interfaces....4
a) Interfaces Utilisateur....4
b) Interfaces Logicielles....4
c) Interfaces de communication....5
d) Bases de données externes....5
3. Contraintes générales de conception....5
II. NORMES , STANDARDS ET OUTILS....5
1. Méthodes de conception....6
2. Environnement et outils de développement....7
3. Conventions de nommage....8
4. Noms des applications et des packages....8
5. Base de données....8
6. Application....9
a) Couche mapping :....9
b) Couche métier....10
c) couche application....10
d) Spring Boot....10
7. Standards de programmation....10
III. CONCEPTION GÉNÉRALE....11
1. Langages utilisés....11
2. Diagramme de déploiement....12
3. Architecture des 5 couches....12
a) Couche Données :....13
b) Couche Mapping :....13
c) Couche Métier :....13
d) Couche Application :....13
e) Couche Présentation :....13
f) L'architecture MVC....14
4. Stratégie de traitement des erreurs et des exceptions....14
a) Stratégie d'exceptions par couches....14
b) Vérification des données saisies....15
c) Sécurisation des accès aux données....15
IV. CONCEPTION DÉTAILLÉE DES COMPOSANTS....16
1. Règles de gestion....16
2. Base de données....17
d) Modèle Conceptuel de Données....17
3. Modèle Logique Relationnel....18
e) Architecture détaillée du composant....19
f) Présentation....20

I. DOMAINE D ' APPLICATION

1. Objectifs

L'objectif de ce projet est de mettre en place un système de facturation électronique qui non seulement permettra de rengrener une estimation de maniéré automatique. Ce système est constitué d'une seule application. Une **application nationale (AN)** installée au siège de la société (Presprint Plc).

Application nationale permettra au personnelle de l'entreprise à réaliser les opérations autre fois manuelle de maniéré automatique de l'arrivée du client jusqu'à la livraison de son produit finale. Cette application doit être utilisable sur l'intranet de la société ou à l'extérieur à l'aide d'un navigateur WEB.

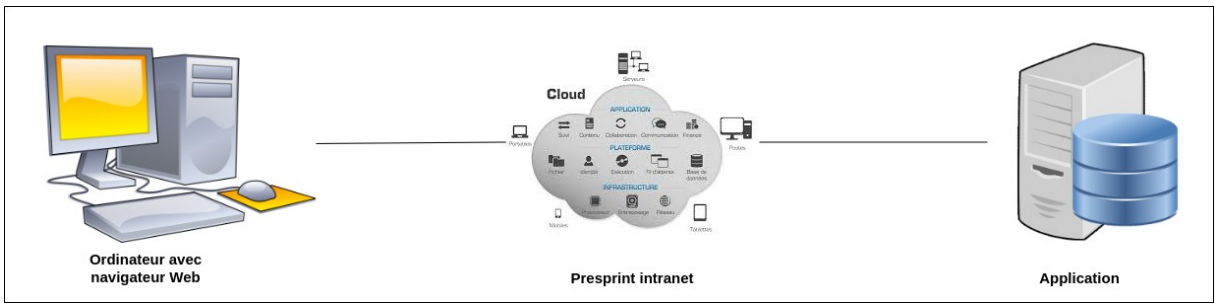


Illustration 1: Architecture du logiciel Billing-system

2. Interfaces

a) Interfaces Utilisateur

Le système contient une application nationale . L'interface de l'application nationale est une interface WEB (via un navigateur WEB tel que Internet Explorer ou Firefox) contenant des formulaires et des tableaux de données regroupés par fonctions, le tout agencé par une navigation par onglets. Cette interface WEB permet l'affichage et la saisie de données. Technologie disponible, taille de l'écran, résolution. Pour plus d'informations sur cette interface, se référer directement à la charte graphique du projet Billing-system.

l'application comporte à son point d'entrée une authentification par mot de passe de l'employé qui va utiliser les services proposés. En fonction de la nature de l'utilisateur, des droits ou des restrictions seront configurés sur l'application.

b) Interfaces Logicielles

Les interfaces logicielles mentionnées ici sont des interfaces internes au système.

- **Interface avec la base de données centrale** : l'application du système Billing-system doit interagir avec la base de données centrale (MySql, installée sur le serveur de base de données) au moyen de pilotes dédiés, de manière synchronisée. Cette base de données rassemble toutes les données manipulées par l'application nationale.

c) Interfaces de communication

Les interfaces de communication mentionnées ici sont des interfaces internes au système.

- **Interface entre le client WEB et l'application** : pour accéder aux données et aux services présentés par l'application nationale, une connexion TCP/IP est nécessaire.

d) Bases de données externes

Aucune base de données externe n'est prévue pour interagir avec notre système. La seule bases de données, mentionnées précédemment, font partie du système à concevoir.

3. Contraintes générales de conception

Plusieurs contraintes provenant de différentes sources sont à prendre en compte dans la phase de conception du système. Ci-dessous, un récapitulatif des contraintes imposées par le maître d'ouvrage dans le cahier des charges :

- L'architecture **Spring** doit être utilisée pour le système, avec l'utilisation des composants Hibernate et Spring Boot.
- La base de données doit être de type MySql pour le système, Cette base de données

doit être synchronisée et disponible à intervalle régulier.

- Le système devra être en français et en anglais.
- La connexion au système Billing-system doit être sécurisées par mot de passe. Chaque catégorie d'utilisateurs disposera de droits spécifiques.

II. NORMES , STANDARDS ET OUTILS

Les documents suivants sont à utiliser en référence avec la lecture de ce document :

- **Le Cahier De Charges** dans sa version finale : il contient les spécifications initiales des exigences du maître d'ouvrage
- **Le Modèle Métier** : il correspond au modèle conceptuel des données, et est accompagné des règles de gestions, des contraintes d'historisation et de cohérence des données.
- **La Charte Graphique** : elle a été établie spécialement pour le projet Billing-sytem
- **Le Dossier Développeur** : il contient les instructions techniques de mise en place de l'environnement de développement et de l'architecture.

Les documents suivants ont été utilisés pour améliorer les concepts introduits dans ce dossier :

- **Cours de Qualité** (William Boher-Coy)
- **Cours de Génie Logiciel Appliqué et de Génie Logiciel** (William Boher-Coy)
- **Cours de JAVA** (Yannick Tisson)
- **Cours de Spring boot** (https://spring.io/projects/spring-boot)

1. Méthodes de conception

Les méthodes de conception sont utilisées afin d'améliorer la qualité de la conception finale.

La méthode de conception MERISE a été utilisée pour mettre en place le Modèle Métier du système, le Modèle Conceptuel de Données (MCD), le Modèle Logique de Données (MLD) pour aboutir enfin au script de génération de la base de données. Ces différents modèles ont été créés grâce à l'environnement de conception Draw.io.

Les recommandations ACAI en terme de modélisation et de conception-réalisation d'applications (documents applicables) ont constitué des références pour les phases d'analyse et de conception du projet.

La **norme IEEE 1471 (2000)** représente également une source de recommandations importante en ce qui concerne l'architecture en 5 couches du système. Elle préconise ainsi l'utilisation intensive de vues et notamment le Modèle-Vue-Contrôleur (MVC – cf. Dossier Développeur) utilisé par la couche « Client».

Les **designs patterns** sont utilisés pour améliorer l'architecture du logiciel. Ce sont des modèles de conception réutilisables qui répondent à des problématiques courantes de conception indépendamment de tout langage.Ces modèles de conception fournissent un support fort pour la mise en œuvre de principes chers à l'approche par objets : la flexibilité, la ré-utilisabilité, la modularité, la maintenabilité…

Concernant l'élaboration du système, un **style de conception ascendante (ou « bottom-up »)** a été choisi. Cette approche permet de s'appuyer sur un modèle métier validé par le maître d'ouvrage, puis de le transférer rapidement en modèle objet. Elle permet également d'intégrer les frameworks et l'architecture en couches, dans l'optique de fournir des composants réutilisables et autonomes.

2. Environnement et outils de développement

Le matériel de développement utilisé est une machine préparée pour chaque développeur, équipée du système d'exploitation Ubuntu et d'une quantité suffisante de mémoire vive (le minimum a été fixé à 1Go pour avoir une qualité de développement acceptable, en partie en raison des nombreux services à exécuter).

Les deux membres de l'équipe de développement exécutent l'application du projet Billing-system sur leurs propres machines. La base de données MySql est située sur une troisième machine personnelle ayant le rôle de serveur central. trois machines sont donc nécessaires pour le développement du projet.

L'outil de développement Eclipse est mis à disposition de l'équipe de développement. Les environnements de développement qui interviennent dans la conception et le développement du système sont Hibernate et Spring.

Une plateforme **Jira** a été mise pour le planning des différentes taches à exécuté par les développeurs par ordre chronologique celles ci contiennent plusieurs statut défini par le chef de projet. Une tache ne pourra être considéré comme achevé que si elle a eu à passer plusieurs test et validé par l'équipe des testeurs .

Pour plus d'informations sur l'ensemble des outils et technologies utilisées dans ce projet, se référez directement au Dossier Développeur ou au Dossier de Gestion de Configuration.

3. Conventions de nommage

Ces conventions concernent les répertoires et dossiers, des entités spécifiques dans les applications du Billing-system (classes, variables, packages, entités de fichiers de configuration) et dans la base de données (tables, champs). Ces conventions respectent les conventions de codage précisées en annexe dans le document ACAI-GuidePersistance-150.

4. Noms des applications et des packages

Le système complet s'appelle « Billing-system » Les packages utilisés dans les sources sont définis comme suit :

- La racine de tout le système est : « Billing-system »
- L'application centrale est située dans le package : « com.ppp»

5. Base de données

Les **tables de la base de données** sont issues de deux catégories de données dans le MCD: les entités (employés, clients, etc.) et les associations (rattachement d'un job à un client, etc.). Chaque entité possède un libellé lisible qui permet de la distinguer clairement (table EMPLOYE au lieu de EMP par exemple). Ce libellé peut

contenir des chiffres, mais ne peut pas contenir d'articles dans le nom (ex : LE_CLIENT), ni de verbes (JOB_RATTACHER_A). Référence : PA52 du Document ACAI-GuidePersistance-150

Les **associations** reliant toujours deux entités au minimum, le libellé des tables correspondantes est une concaténation des deux entités, ce qui donne par exemple « CLIENT_JOB pour la table correspondant à l'association rattachant un client à un Job. Étant donné que plusieurs associations peuvent exister entre 2 tables, on ajoutera alors au libellé de la table le nom caractérisant cette association (RATTACHEMENT, CORRESPONDANCE, etc.). Les noms des tables ne devront jamais dépassés 30 caractères.

Concernant **les champs de la base de données,** les noms des colonnes ont les mêmes règles que les libellés des tables. Le nom de chaque colonne d'une table commence par les 3 premières lettres du nom de la table (hors préfixe). Ceci permet d'identifier très clairement les colonnes de clé étrangère : elles ne commencent pas par les 3 mêmes lettres.

Un **index sur une clé primaire** est créé automatiquement et portera le nom de la clé primaire : PK_nom de la table. Les index sur les clés étrangères seront préfixés de FK. On aura FK_nom de la référence.

Concernant **les requêtes SQL** présentes dans les fichiers sources, elles doivent être écrites de la façon suivante : lettres majuscules pour les mots SQL (ex. : SELECT, UPDATE, TO_CHAR) et lettres minuscules pour les noms des objets sur lequel porte la requête (noms des tables, champs, variables, etc.).PA54 du Document ACAI-GuidePersistance-150.

6. Application

Au sein du système, il convient de nommer chaque couche de façon prédéfinie. En effet, chaque couche est stockée dans un package différent :

- **la couche physique** n'est pas représentée car il s'agit de la base de données
- **la couche mapping** est stockée dans les packages : com.ppp.user.model et com.ppp.user.model.dto
- **la couche métier** est stockée dans le package : com.ppp.user.service.impl

- **la couche application** est stockée dans le package : com.ppp.user.controllers
- **la couche présentation** est stockée dans un dossier séparé webapp contenant des jsp et les ressources associées

a) Couche mapping :

Au sein de la couche mapping, plusieurs conventions de nommage sont à déterminer. On distingue deux types de fichiers : les fichiers de mapping et les classes d'entités métiers . Les classes de mapping développées dans ce projet utilisent le mapping généré par Hibernate. Ces classes d'accès aux données (Data Access Object) sont nommées suivant le schéma <NomDonneesAccedees>DTO.

b) Couche métier

Au sein de la couche métier, plusieurs conventions de nommage sont également à mettre en place. Tout d'abord, les fichiers sont organisés par entités, les entités étant elles-mêmes organisées par groupes d'entités selon l'axe de fonctionnalité (job, employés, clients etc.). Chaque entité est alors représentée par 2 fichiers principaux (d'autres fichiers complémentaires peuvent s'ajouter) :

- les managers, nommés **<NomService>**
- les interfaces associées, implémentées par les managers: **<NomServiceImpl>**

c) couche application

La couche application est constituée des différents contrôleurs permettant d'implémenter chaque service demandé. Ces contrôleurs sont organisés par axes de fonctionnalités (à l'image de la couche métier). Tous les contrôleurs sont nommés suivant le model **<Nom>Controller.**

d) Spring Boot

Spring boot définit des objets et leurs dépendances dans un fichier de configuration xml. Les objets qui font référence aux classes crées pour représenter les différentes

couches sont nommées suivant le model suivant **nomClasse** pour une classe **NomClasse**.

7. Standards de programmation

L'équipe de développement suit un ensemble de conventions de codage qui permettent une homogénéisation des sources :

- **Les commentaires** doivent être rédigés pour chaque classe et chaque méthode en respectant la norme JavaDoc, afin de permettre à toute personne entrant dans le projet de reconnaître l'utilité, les entrées et les sorties de ces entités.
- **Les conventions de codage Java** utilisées sont celles recommandées par les ACAI (cf. Guide de conception-réalisation).
- **Les conventions de codage SQL** sont également celles recommandées par les ACAI (cf. Guide de la persistance).
- **Les conventions approuvées par le W3C** sont également appliquées dans le cadre du développement WEB (HTML et CSS)

III. CONCEPTION GÉNÉRALE

1. Langages utilisés

Voici la liste des différents langages utilisés dans le projet **Billin-sytem**

- **SQL** pour les scripts de création de la base de données (création de tables et insertions de tuples dans la BD).
- **PL/SQL** pour les déclencheurs.
- **HQL** (langage à requêtes de Hibernate, similaire à SQL)
- **Java 8**
- **JSP** pour le développement des vues dans.
- **HTML, CSS et Javascript** pour le contenu des vues JSP
- **XML** pour les fichiers de configuration Spring boot, Hibernate et Tomcat.

2. Diagramme de déploiement

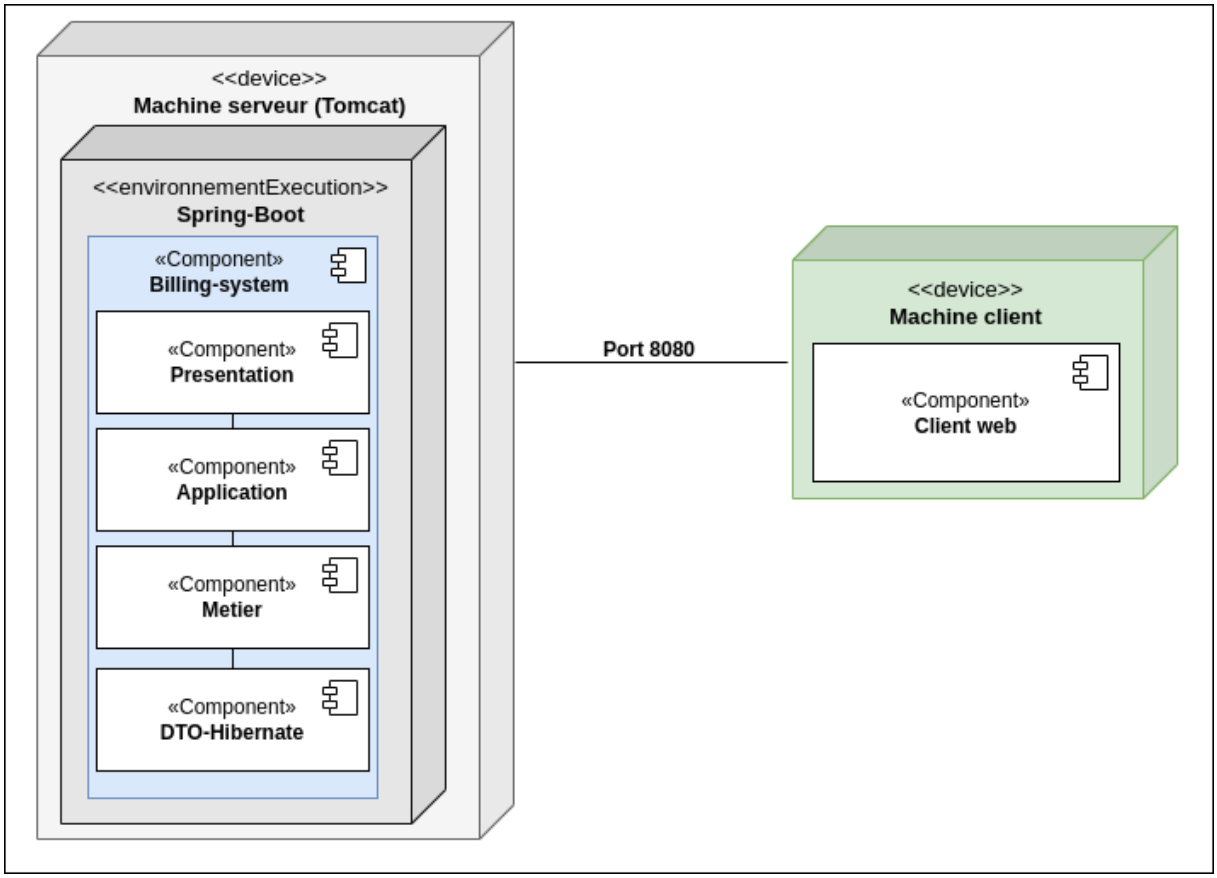


Illustration 2: Diagramme de reploiement

3. Architecture des 5 couches

Le système est divisée en cinq couches de fonctionnalités, totalement autonomes les unes des autres, et communiquant par un système de file : chaque couche ne dialogue qu'avec les couches voisines supérieure et inférieure. Toutes les couches doivent agir de façon transparente les unes des autres. La séparation des couches est la suivante :

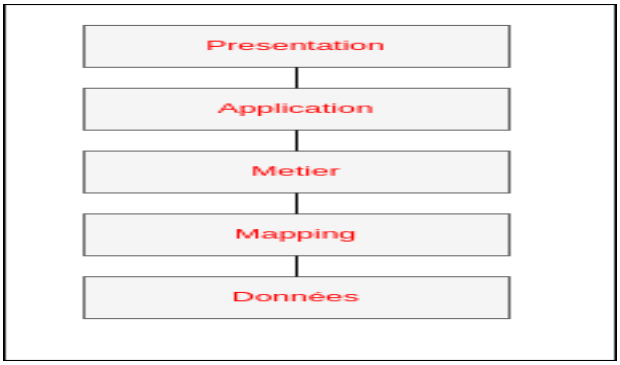


Illustration 3: Diagramme des 5 couches

a) **Couche Données :**

Cette couche contient les données physiques stockées dans la base de données MySql.

b) **Couche Mapping :**

Cette couche contient l'implémentation des accès à la base de données afin de la masquer à la couche métier. Elle est entièrement gérée par Hibernate.

c) **Couche Métier :**

Cette couche contient les objets métiers de l'application. Il existe un objet par fonctionnalité de l'application.

d) **Couche Application :**

Cette couche contient la partie fonctionnelle de l'application. Elle s'appuie sur les objets métier pour réaliser les actions sollicitées par l'utilisateur par l'intermédiaire de la couche présentation. Elle est en charge de vérifier la validité des requêtes de la couche présentation. Il existe un contrôleur par fonctionnalité de l'application. Le schéma utilisé est le MVC (Modèle Vue Contrôleur), mis en œuvre en utilisant Spring MVC.

e) **Couche Présentation :**

Cette couche représente les interfaces qui permettent de présenter les contenus générés

par la couche présentation, grâce à des vues dynamiques (JSP).

f) L'architecture MVC

Les couches présentation et application s'appuient sur l'architecture « Modèle-Vue-Contrôleur » qui permet de séparer le fonctionnel de l'interface. Cette architecture est réalisée par une conjonction d'un contrôleur et d'un nombre quelconque de pages JSP (les vues) qui offrent le rendu à l'utilisateur. Les données calculées par le contrôleur et fournies aux vues pour être affichées sont les modèles. Le contrôleur s'appuie sur les couches inférieures pour obtenir ces données.

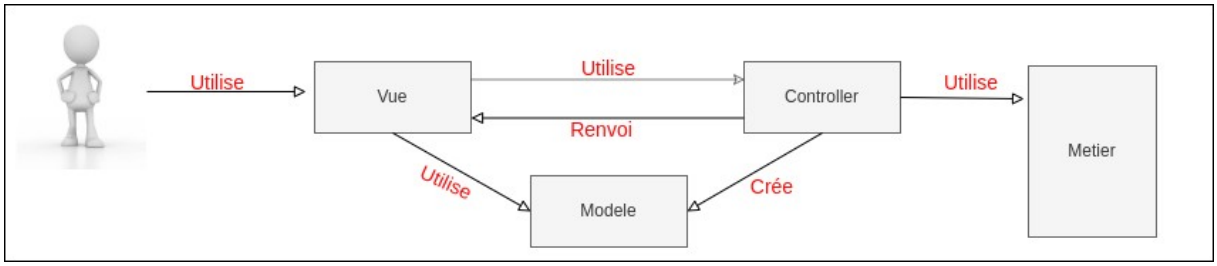


Illustration 4: Architecture du MVC

Plus précisément, l'utilisateur sollicite une action par l'intermédiaire d'une Vue. Cette action est transmise au contrôleur, qui en vérifie la validité et s'appuie sur la couche métier pour l'effectuer. Enfin le contrôleur renvoi sur la vue correspondant à la demande de l'utilisateur.

4. Stratégie de traitement des erreurs et des exceptions

Dans ce paragraphe sont exposées les différentes stratégies mises en place pour gérer les erreurs, i.e. empêcher l'application de s'effondrer sur n'importe quelle erreur et rendre les erreurs compréhensibles pour l'utilisateur.

a) Stratégie d'exceptions par couches

Afin de permettre une gestion localisée des exceptions, il a été décidé que chaque couche prenne en compte. Ainsi, la couche mapping est en charge des exceptions lors des

accès à la base. La couche contrôleur est en charge des exceptions dues au mauvais formatage des données et de transmettre des messages d'erreur textuels à la couche présentation pour l'utilisateur.

b) Vérification des données saisies

Une des principales sources d'erreurs dans les applications est due à des saisies erronées de la part de l'utilisateur. Ces erreurs peuvent être de plusieurs niveaux: erreurs de formatage, non suivi des règles métier, contraintes de base de données. Les différents niveaux de vérifications de données sont les suivants :

- **couche contrôleur** : les formulaires sont validés. On peut alors vérifier que les données ne sont pas absentes, ou mal formatées.
- **couche métier** : une fois les valeurs saisies validées, la prochaine étape est de vérifier l'adéquation métier de ces données.
- **couche mapping** : cette couche se charge de vérifier toutes les règles de gestion implémentées dans la base de données.

c) Sécurisation des accès aux données

Les données sont accessibles au travers de différents écrans, offrant les différentes fonctionnalités de l'application. Les utilisateurs de la partie administration doivent s'identifier pour y accéder. De plus un groupe est définit pour chacun d'eux. Pour chaque groupe, les fonctionnalités accessibles sont définies dans la base de données. L'application est développée pour prendre en compte les fonctionnalités accessibles à l'utilisateur courant. Pour cela elle n'affiche que les écrans autorisés à l'utilisateur. De plus certaines portions de formulaires ou de menus peuvent être masquées.

Pour empêcher l'accès aux écrans non autorisés, en plus de les masquer, un contrôleur spécifique, appelé intercepteur vérifie qu'un utilisateur peut bien y accéder pour chaque requête. Les droits d'accès sont éditables par l'administrateur qui dispose de tous les droits sur tous les écrans de l'application.

IV. CONCEPTION DÉTAILLÉE DES COMPOSANTS

1. Règles de gestion

Les règles de gestion décrivent la nature des relations entre les entités d'un système d'information. L'ensemble de ces règles permet de définir un système correspondant à une problématique métier précisément adaptée aux besoins du client. Ces règles de gestion sont utilisées directement dans le modèle métier : chaque règle correspond à une relation entre 2 (ou plusieurs) entités. Les entités métiers sont indiquées en gras.

- R1 : Un **client** peut avoir plusieurs **job** . Un **job** appartiens à un **client** .
- R2 : **Un job_type** peut appartenir à plusieurs **job** . Un **job** a un seul **job_type**
- R3 : Un **utilisateur** peut avoir plusieurs **job_tracking** . Un **job_tracking** appartient à un seul **utilisateur** .
- R4 : Un **paper_type** appartiens à plusieurs **job_paper** . Un **job_paper** a un seul **paper_type** .
- R5 : Un **job_paper** appartient à un **job** . Un **job** a plusieurs **job_paper** .
- R6 : Une **printing_machine** peut imprimer plusieurs **job_paper** . Un **job_paper** peut être imprimé par un **printing_machine** .
- R7 : Un **job_paper** peut avoir plusieurs **job_pricing** . Plusieurs **job_pricing** a un **job_paper** .
- R8 **:** Un **price_element** peut avoir plusieurs **job_pricing** . Plusieur **job_pricing** appartiennent à un **price_element**.
- R9 : Un **job_operation_option** peut avoir plusieurs **price_element** . Plusieurs **price_element** appartiennent à un seul **job_operation_option**.
- R10 : Un **job_operation_option** peut avoir plusieurs **job_activity_option** . Plusieurs **job_activity_option** appartiennent à un **job_operation_option** .
- R11 : Un **job_activity_option** peut appartenir à plusieurs **job_activity** . Plusieurs **job_activity** appartiennent à un seul **job_activity_option .**
- R12 : Un **job** a plusieurs **job_activity** . Plusieurs **job_activity** peuvent appartenir à

un **job .**

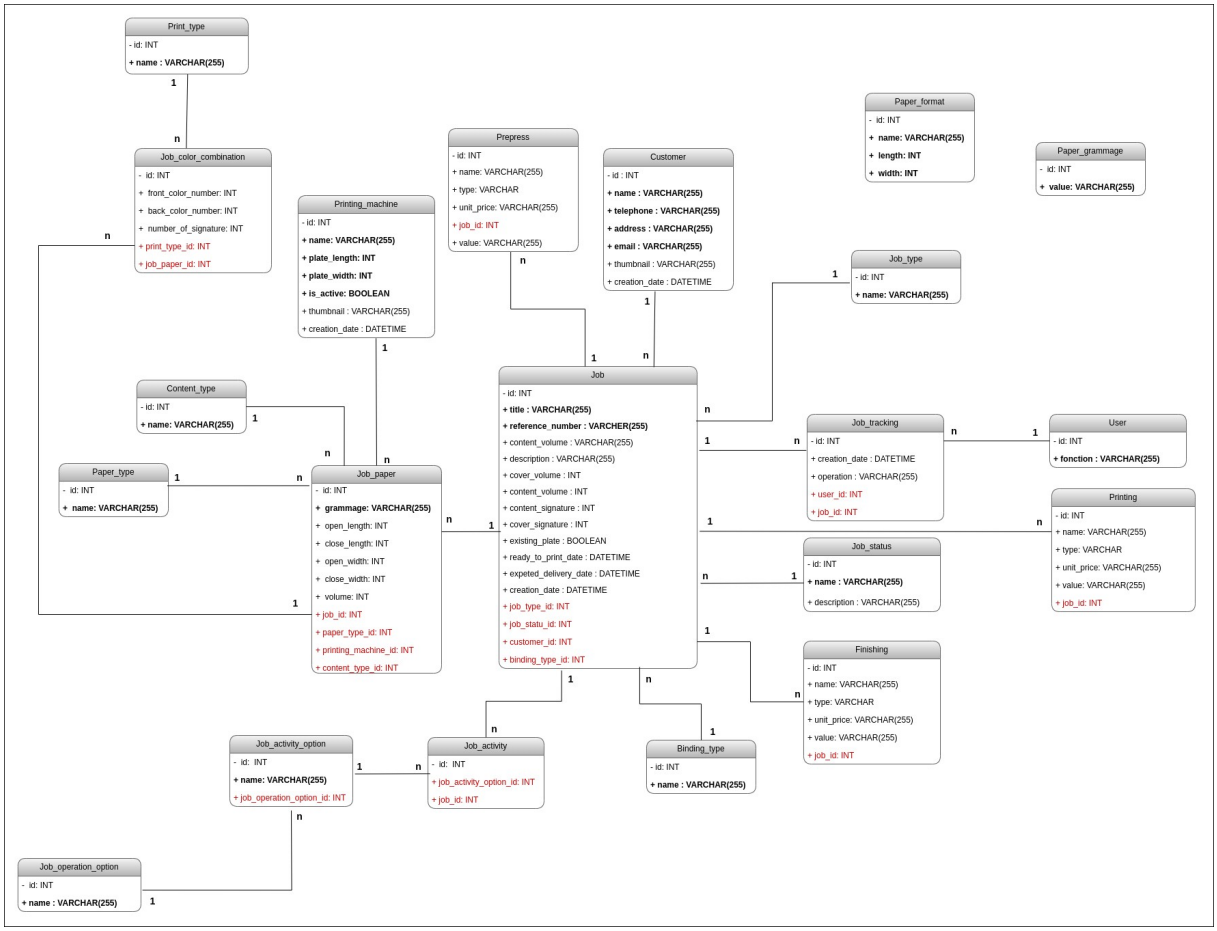
- R13 : Un **job** a plusieurs **binding_type** . Plusieurs **binding_type** appartiennent à un job.
- R14 : Un **job_status** peut concerner plusieurs **job .** Plusieurs **job** peuvent avoir un **job_status** .
- R15 :Un **job** peut avoir plusieurs **job_tracking** . Plusieurs **job_tracking** peuvent appartenir à un **job** .
- R16 : Un **job** peut avoir plusieurs **job_color_combinaition** . Plusieurs **job_color_combinaition** peuvent appartenir à un **job**.
- R17 : Un print_type peut avoir plusieurs **job_color_combinaition .** Plusieurs **job_color_combinaition** appartiennent à **unprint_type** .
- R18 : Un **content_type** peut avoir plusieurs **job_color_combinaition** . Plusieurs **job_color_combinaition** appartiennent à un **content_type** .
- R19 : Un **content_type** appartient à un **job_paper** . Plusieurs **job_paper** appartiennent à un **content_type** .

2. Base de données

d) **Modèle Conceptuel de Données**

La représentation du système d'information a été réalisé à l'aide du modèle métier et de la méthode MERISE. Cette méthode nous a permis de réaliser le Modèle Conceptuel de Données ou MCD, qui fait le lien entre le modèle métier et le modèle logique. Le MCD est un modèle dit « entités-relations » qui modélise l'ensemble des règles de gestion et des contraintes du système. Il correspond globalement au modèle métier, en détaillant le contenu des entités métiers (attributs).

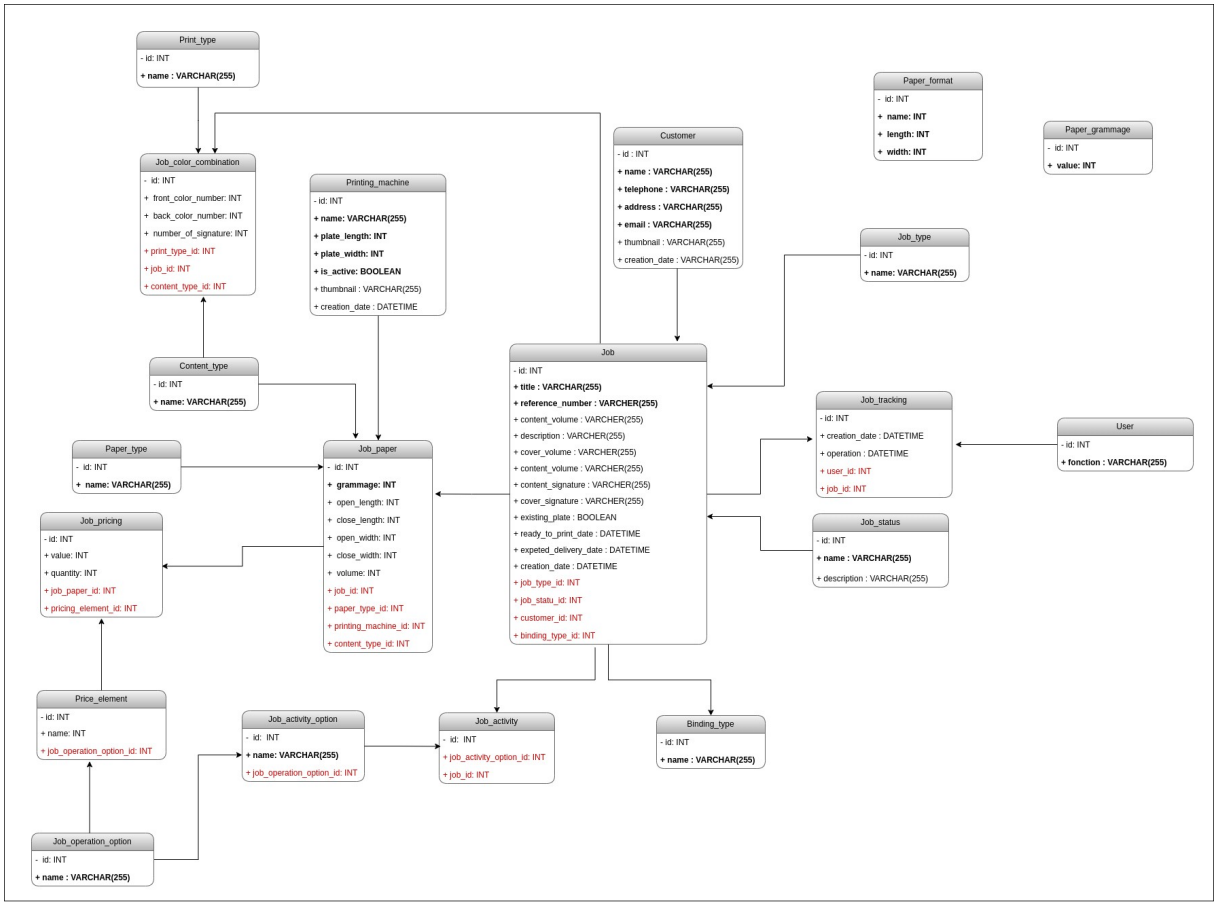
La construction du MCD a été réalisé à partir des spécifications du cahier des charges, du modèle métier et des réunions avec le maître d'ouvrage. Celui-ci a été généré grâce au logiciel **draw.io** .



3. Modèle Logique Relationnel

Le MLR est le modèle logique correspondant au MCD. Il représente les tables et les colonnes de la base de données, le schéma de la base de données relationnelles. Chaque entité du MCD correspond à une table, de même que les associations 0..n # # 0..n et les associations contenant des attributs .

Le paramétrage consiste aussi à définir le type de données pour chaque champ. Les types de chaque champs sont définis à partir des informations contenues dans le cahier des charges. On utilisera ainsi des INT pour les entiers (la taille du NUMBER variant en fonction de la précision attendue, ou de la quantité d'enregistrements attendue dans la table) , des CHAR(32) ou des VARCHAR2(255) pour les textes, des DATE pour les dates, etc.



e) **Architecture détaillée du composant**

L'application se divise en deux parties : une partie destinée aux différents staffs techniques de Presprint, permettant principalement d'effectuer les opérations sur un job ; et une partie administration, destinée à l'administrateur et offrant toutes les fonctionnalités de gestion définies dans le cahier des charges .

Pour mettre en œuvre cette architecture nous avons utilisé :

- Oracle 10g XE pour la couche Données ;
- Apache Commons DBCP pour le pool de connections ;
- Hibernate pour le mapping de la base de données ;
- Spring IOC pour les liens entre les différentes couches ;

- Spring MVC pour la mise en place du MVC ;
- JSTL (JavaServer Pages Standard Tag Library) pour l'affichage des données dans les JSP.