|  |
| --- |
| Emilie Bonnin & Morgane LOPVET – Apping2 2014 |
| Rapport du projet J2EE |
| Intranet de la Poste: Gestion des erreurs de caisse |

|  |
| --- |
| Morgane LOPVET  [Choisir la date] |

Table des matières

[1. Introduction 2](#_Toc342658040)

[1.1. Présentation du projet 2](#_Toc342658041)

[1.2. Equipe 2](#_Toc342658042)

[1.3. Ce qui a été fait 2](#_Toc342658043)

[1.4. Technologies utilisées 2](#_Toc342658044)

[2. Réalisation 2](#_Toc342658045)

[2.1. Conception 2](#_Toc342658046)

[2.1.1. Modèle conceptuel de données 2](#_Toc342658047)

[2.1.2. Modèle physique de données 3](#_Toc342658048)

[2.1.3. Diagramme d’activités UML 3](#_Toc342658049)

[2.1.4. Diagramme de classes UML 4](#_Toc342658050)

[2.2. Développement 5](#_Toc342658051)

[2.2.1. Base de données 5](#_Toc342658052)

[2.2.2. Classes métier 5](#_Toc342658053)

[2.2.3. Servlets et JSP 5](#_Toc342658054)

[3. Mode d’emploi 6](#_Toc342658055)

[3.1. Compilation, déploiement et exécution 6](#_Toc342658056)

[3.2. Exemples 6](#_Toc342658057)

[3.3. Tests 6](#_Toc342658058)

[4. Conclusion 6](#_Toc342658059)

# Introduction

## Présentation du projet

Le projet consiste en un intranet permettant la gestion des erreurs de caisse dans les bureaux de Poste. Il est destiné aux agents de la Poste. Il doit permettre de signaler et de régulariser totalement ou partiellement des erreurs de caisse ainsi que d’en assurer le suivi. La gestion des droits est également indispensable et l’authentification est obligatoire.

## Equipe

Notre équipe se compose de Emilie Bonnin et Morgane Lopvet, toutes deux apprenties et suivant une formation en ingénierie informatique à Epita.

Morgane travaille chez Eureva, société pour laquelle elle développe des applications mobiles, notamment utilisant la reconnaissance d’image.

Emilie…

## Ce qui a été fait

Toutes les fonctionnalités requises dans le sujet ont été implémentées. C’est-à-dire :

* Gestion des droits
* Déclaration d’une erreur de caisse
* Consultation des erreurs de caisses
* Régularisation partielle ou totale d’une erreur de caisse
* Consultation des régularisations des erreurs de caisse
* Calcul des bilans des erreurs de caisse journaliers et périodiques
* Envoi d’un mail d’avertissement en cas de dépassement d’un certain montant du bilan (en déficit ou excédent)

Nous avons également mis en place des fonctionnalités de recherche pour les erreurs de caisse.

## Technologies utilisées

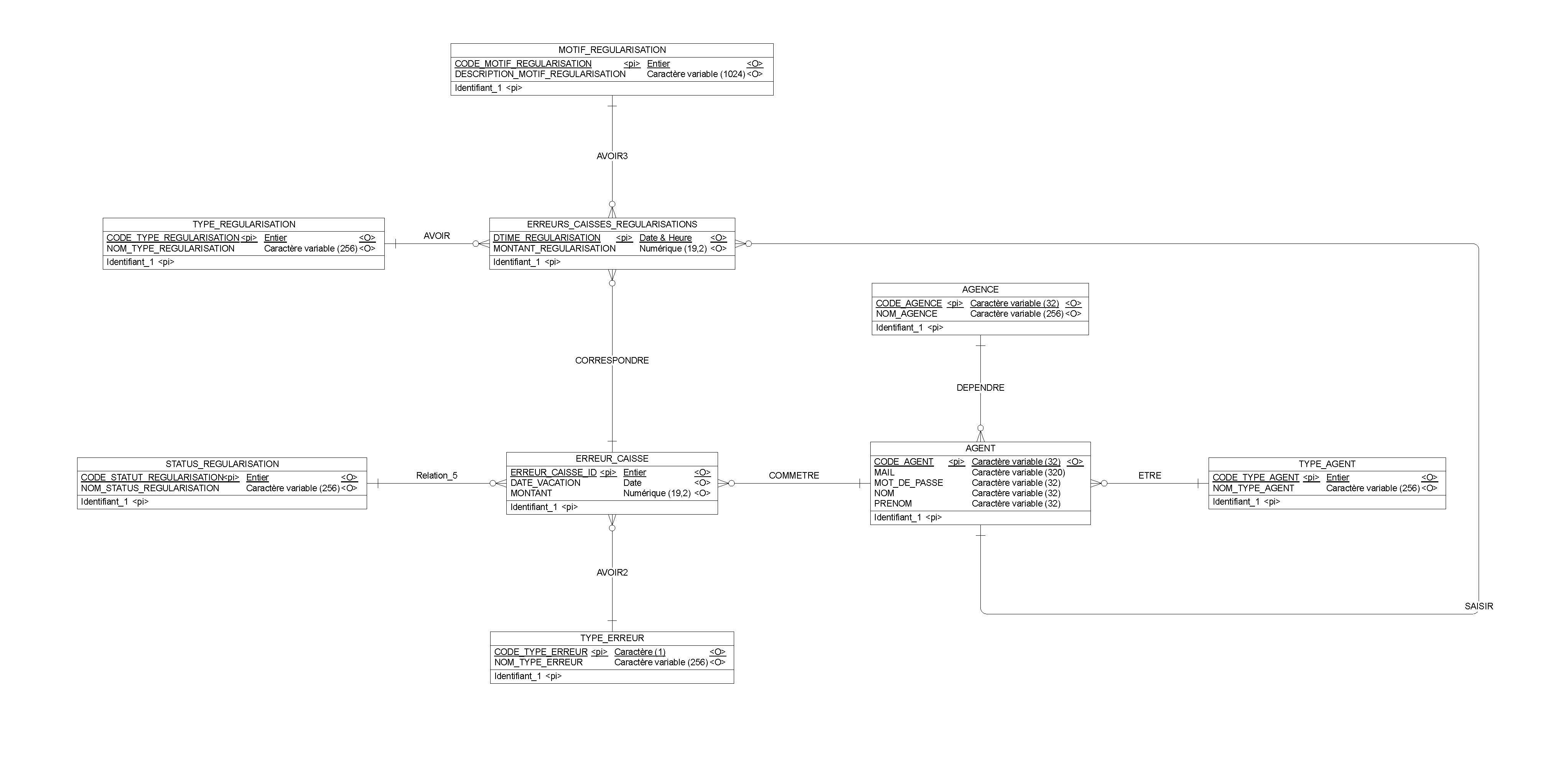
Nous avons choisi d’utiliser une base de données Oracle accessible au code java grâce à la plateforme JDBC. Nous nous sommes servies des servlets ainsi que de pages jsp. La technologie JSTL ainsi que les EL nous ont permis de faire communiquer le contrôleur avec la vue. Nous avons respecté un modèle MVC. Le serveur Tomcat a été utilisé pour déployer l’application. L’API Java mail est également de la partie. Du côté de l’affichage, nous avons utilisé des ccs et du JavaScript via les bibliothèques JQuery et TableSorter.

# Réalisation

## Conception

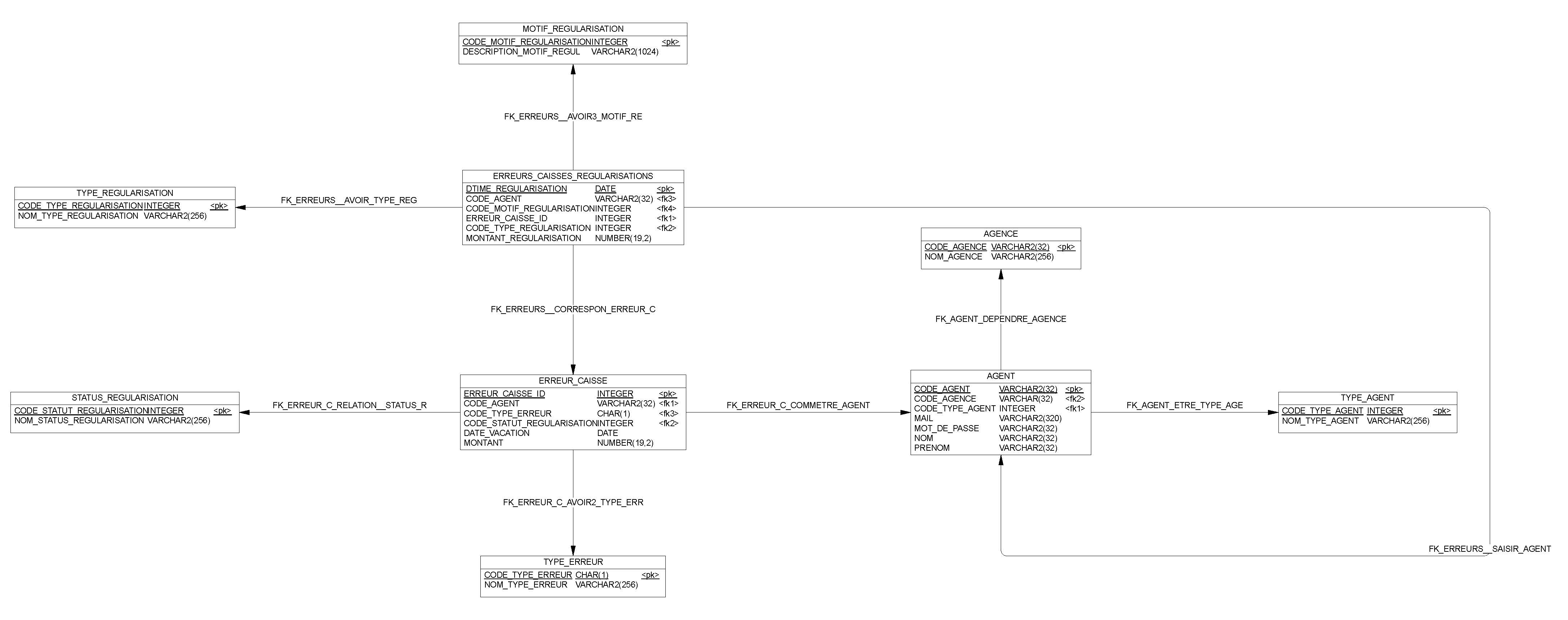
### Modèle conceptuel de données

Après avoir étudié le sujet, nous avons réalisé ce modèle conceptuel de données. Afin d’avoir une organisation propre et permettre une évolution des fonctionnalités, nous avons séparé au maximum les différentes entités.



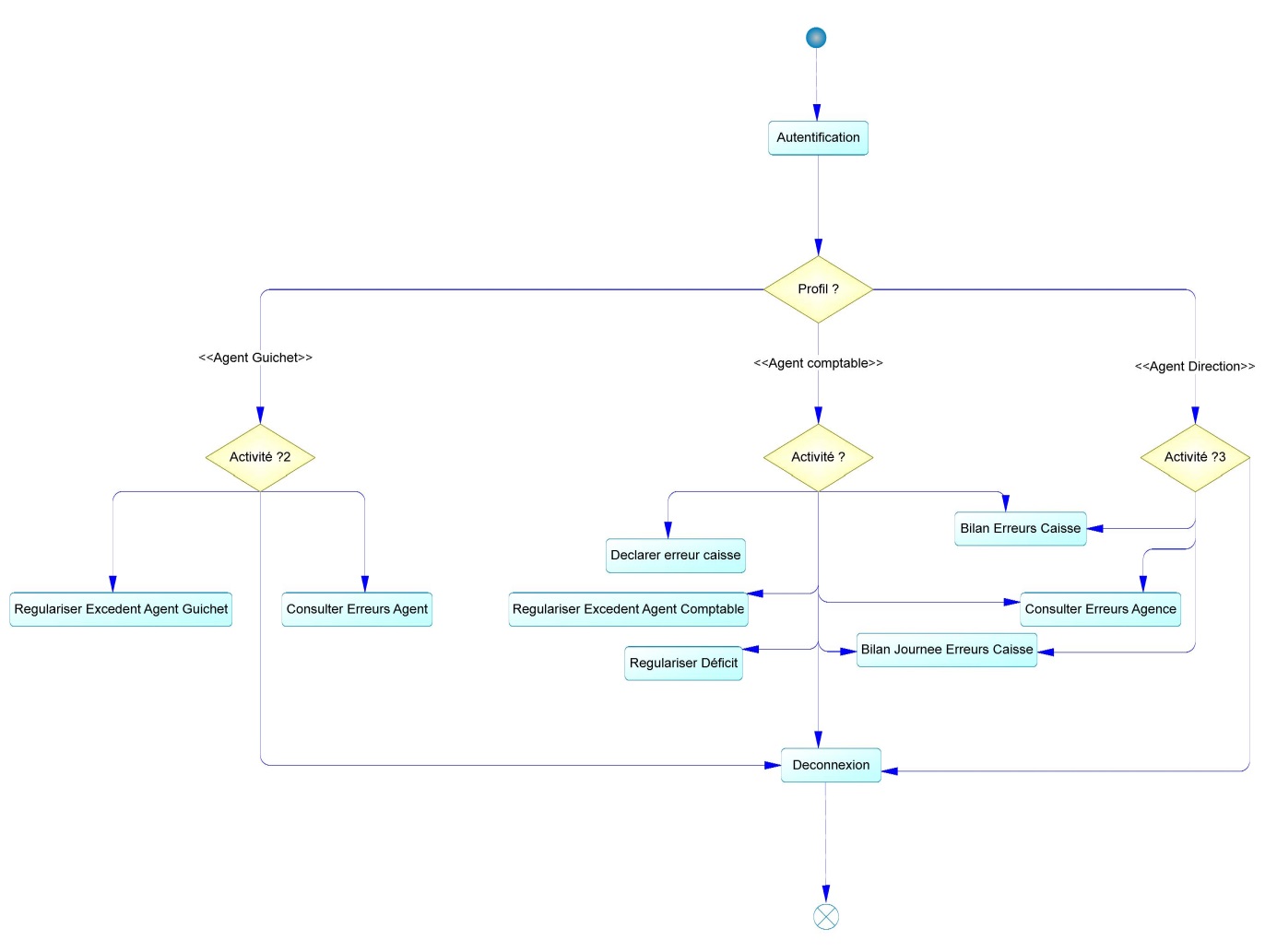
### Modèle physique de données

Le modèle conceptuel de données nous a alors permis de concevoir le modèle physique de données. C’est ce modèle qui représente notre base de données. Le moindre détail, la moindre contrainte a été indiquée dans ce modèle afin que le logiciel PowerAMC génère directement le code SQL de création de la base de données. Nous avons notamment utilisé les séquences et les triggers pour gérer l’auto-incrémentation de certains identifiants, comme les ID\_ERREUR\_CAISSE.



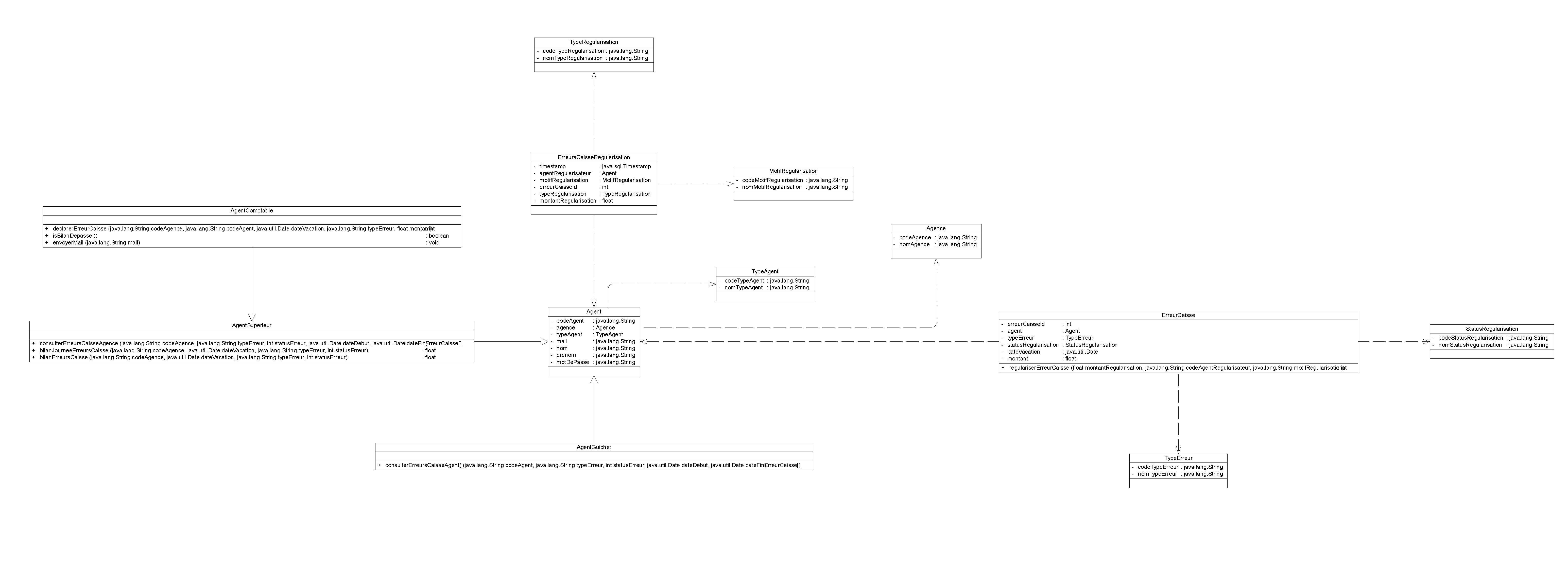
### Diagramme d’activités UML

A ce stade de la conception, nous avons voulu avoir une vision plus précise des fonctionnalités à implémenter ainsi que des différents chemins de navigation possibles. A cette fin, nous avons conçu le diagramme d’activités UML suivant.



### Diagramme de classes UML

Une fois la base de données mise en place et le digramme d’activité défini, nous avons réfléchis au diagramme de classes UML. Seules les classes métier ont été spécifiées mais quelques classes utilitaires ainsi que des DAO (uniques classes accédant directement à la base de données) ont été implémentées.



Afin de gérer les différents agents ainsi que leurs droits, nous avons décidé de créer une classe agent de laquelle hérite des agents de type spécifique en fonction des fonctionnalités qui lui sont accessibles. Ainsi, les agents de direction et comptables ont accès à des fonctionnalités communes mais l’agent de direction n’a aucune fonctionnalité exclusive. La classe AgentSupérieur réunit dont les fonctionnalités communes aux agents comptables et de direction tandis que les agents de guichet et comptables ont chacun leur propre classe implémentant les fonctionnalités qui leurs sont réservées.

## Développement

### Base de données

Nous avons commencé par gérer les accès à la base de données.

Nous avons choisi de cloisonner les appels à la base de données dans des classes spécifiques (package BDD) afin de garder une structure cohérente et éviter les erreurs. Ces classes permettent donc d’exécuter les requêtes classiques (insert, update, select \*) ainsi que d’autres requêtes spécifiques à certaines méthodes des classes métiers, comme celle qui calcule le bilan d’un bureau de Poste.

Nous avons également mis en place un Singleton pour la connexion à la base de données pour s’assurer qu’une seule connexion était ouverte à la fois.

### Classes métier

Les classes métiers ont été implémentées en adéquation avec le diagramme de classes UML présenté ci-dessus. La plupart des méthodes étaient relativement simples à mettre en place. Mais la méthode chargée régulariser une erreur de caisse a nécessité une phase de réflexion plus importante. Son code est bien commenté.

### Servlets et JSP

Une fois les classes métiers finalisées, nous avons mis en place les jsp qui fonctionnent toujours avec une servlet. La servlet s’occupe du traitement en appelant les classes métiers (c’est le contrôleur) tandis que les jsp affichent le résultat.

#### Taglibs

Nous avons utilisé deux taglib de la JSTL (core et format) afin de clarifier l’écriture des jsp. Nous avons définit ces taglib dans le fichier « taglibs.jsp » et l’avons déclaré dans le fichier « web.xml » afin qu’elles soient automatiquement incluses dans toutes les jsp de l’application.

#### 2.2.3.2. Gestion des droits

La gestion des droits a été grandement facilitée par notre choix de modèle. Nous avons choisi de mettre un agent en session dès le login validé et avons écrit deux fonctions utilitaires qui se chargent de vérifier que l’agent est bien connecté et qu’il a bien les droits requis pour voir une page ou effectuer une opération. Ces deux fonctions sont appelées dans toutes les servlets. Dans ce contexte, la mise en place de filtres au niveau des jsp ne nous a pas paru primordiale.

# Mode d’emploi

## Compilation, déploiement et exécution

Pré-requis :

* Avoir Tomcat 6 installé
* Oracle Application Express 11g installé avec le mot de passe « a ».
* Créer un profil utilisateur nommé « INTRAPOSTE » avec pour mot de passe « a » dans Oracle Application Express

Exécuter le script « base.sql » dans Oracle Application Express. On pourra initialiser la base avec des données directement depuis l’application, en cliquant sur le lien « Initialiser la base » depuis la page d’identification.

Aller ensuite dans le Gestionnaire d'applications WEB Tomcat et téléverser le ficher war fourni en cliquant sur « déployer ». Cliquer sur le chemin de l’application.

## Exemples

## Tests

Nous avons mis en place des tests unitaires avec JUnit pour les classes métier afin de nous assurer de leur bon fonctionnement et éviter les régressions suite à des éventuels changements dans ces classes.

Nous avons également défini une liste de tests afin de vérifier le bon fonctionnement de l’application dans son ensemble :

# Conclusion

Grâce à notre organisation et notre cohésion, nous avons pu aller au bout de ce projet, en ajoutant même certaines fonctionnalités non requises par le sujet.

Si nous avions déjà développé un projet en J2EE l’année dernière, ce projet a été l’occasion de découvrir comment fonctionnaient les interactions entre jsp et servlets sans passer par un Framework tel que Strut2s. Cela a également été l’occasion de découvrir le moteur de base de données Oracle donc ne nous étions jamais servies.

On pourrait compléter l’application en offrant une gestion des utilisateurs (création, mise à jour, récupération de mot de passe). Les classes métiers existent déjà et les appels à la base de données pour effectuer ces opérations sont déjà implémentés. Il suffirait donc d’écrire deux nouvelles jsp avec leur servlet respectif, une pour la création/modification d’un agent, l’autre pour la récupération du mot de passe.