



Institut Supérieur d'Informatique,  
de Modélisation et de leurs Applications  
Campus des Cézeaux  
BP 10125  
63173 AUBIÈRE CEDEX

RAPPORT DE PROJET DE 2<sup>ÈME</sup> ANNÉE  
FILIERE SYSTÈME D'INFORMATION DE PRODUCTION ET D'AIDE À LA DÉCISION

# SYSTÈME DE GESTION DE BILLETS ÉLECTRONIQUES (LIGHTPAW)

---

Présenté par :

- **AHTUNE Gabriel**
- **BELLOGE Styvens**

Tuteur : **DUHAMEL Christophe**

Durée : 100 heures



Institut Supérieur d'Informatique,  
de Modélisation et de leurs Applications  
Campus des Cézeaux  
BP 10125  
63173 AUBIÈRE CEDEX

RAPPORT DE PROJET DE 2<sup>ÈME</sup> ANNÉE  
FILIERE SYSTÈME D'INFORMATION DE PRODUCTION ET D'AIDE À LA DÉCISION

# SYSTÈME DE GESTION DE BILLETS ÉLECTRONIQUES (LIGHTPAW)

---

Présenté par :

- **AHTUNE Gabriel**
- **BELLOGE Styvens**

Tuteur : **DUHAMEL Christophe**

Durée : 100 heures

# Remerciements

Nous tenons à remercier notre chef de projet Christophe DUHAMEL pour ses conseils tout au long du projet et ses suggestions de généralisations du produit.

# Résumé

Les associations à but non lucratifs ont besoin d'un système de gestion de billets électronique afin de faciliter la vente de billet destinée à l'accès à des évènements.

Le projet consistait donc à développer un système de gestion de billet efficace, facile et peu onéreux à mettre en place. Puis par la suite il s'est généralisé à la gestion de tout éléments faisant l'objet d'une vente ; les billets se sont généralisé a des cartes à crédit.

**Mot-clés :** Billet électronique, Reseaux de distribution, Code Barre, PHP, MySQL

# Abstract

The non-profit organizations needs a system for managing electronic ticket in order to facilitate the sale of tickets for access to events.

Firstly, The project has been to developped a management system ticket efficient, easy and inexpensive to implement. And subsequently it was extended to the management of all tradable items, the tickets were a widespread to credit cards.

**Keywords :** electronic ticket, distribution network, Bar code, PHP, MySQL

# Table des matières

Remerciements

Résumé

Abstract

Table des figures

Glossaire

<b>Introduction</b>	<b>1</b>
<b>1 Modélisation du domaine</b>	<b>3</b>
1.1 Définitions des acteurs . . . . .	3
1.1.1 Les acteurs en présence . . . . .	3
1.1.2 Les acteurs modélisés . . . . .	3
1.2 Définitions des interactions . . . . .	4
1.2.1 Les ventes traditionnelles . . . . .	5
1.2.2 Le modèle de vente . . . . .	5
1.3 Définition des rôles . . . . .	6
1.3.1 Administrateur . . . . .	6
1.3.2 Opérateur Vendeur . . . . .	6
1.3.3 Opérateur Caissier . . . . .	6
<b>2 Conception</b>	<b>7</b>
2.1 Analyse Fonctionnelle . . . . .	7
2.1.1 Contrôle des Transactions . . . . .	7
2.1.2 Authentification des Utilisateurs . . . . .	7
2.2 Choix Architecturaux . . . . .	9
2.2.1 Architecture du serveur . . . . .	9
2.2.2 Architecture des interfaces . . . . .	10
2.2.3 Architecture du système de stockage . . . . .	11
2.3 Organisation des entités . . . . .	11
2.3.1 Utilisateur . . . . .	11
2.3.2 Transaction . . . . .	12
2.3.3 Article et Billet . . . . .	12
2.4 Gestion de la sécurité . . . . .	13
2.4.1 Séparation des pouvoirs . . . . .	13
2.4.2 Séparation des infrastructures . . . . .	13
2.4.3 Archivage de toutes les transactions . . . . .	14
2.4.4 Protection des scripts . . . . .	14
<b>3 Réalisation</b>	<b>15</b>

3.1	Réalisations relatives aux bases de données . . . . .	15
3.1.1	Gestion des accès concurrentiels . . . . .	15
3.1.2	Sécurisation des requêtes base de données . . . . .	16
3.2	Réalisations relatives à l'application serveur . . . . .	17
3.2.1	Les outils utiles à l'application . . . . .	18
<b>4</b>	<b>Résultats et prolongement</b>	<b>20</b>
4.1	Présentation du produit . . . . .	20
4.2	Prolongement . . . . .	21
4.2.1	L'existant . . . . .	21
4.2.2	Les ajouts futurs . . . . .	21

**Conclusion**

**Bibliographie**

**Annexes**

# Table des figures

1.1	Schémas des acteurs d'une organisation . . . . .	4
1.2	Diagramme UML de communication représentant une vente traditionnelle . . . . .	5
1.3	Diagramme UML de communication représentant le modèle de vente . . . . .	5
1.4	Diagramme UML de cas d'utilisation : les rôles des acteurs . . . . .	6
2.1	Diagramme de la fonctionnalité Réalisation des transactions . . . . .	8
2.2	Diagramme de la fonctionnalité Authentification des utilisateurs . . . . .	8
2.3	Diagramme de la fonctionnalité Contrôle des transactions . . . . .	9
2.4	Diagramme UML de communication du patron Modèle-vue-contrôleur . . . . .	10
2.5	Interfaces . . . . .	10
2.6	Utilisateurs et acteurs . . . . .	12
2.7	Diagramme UML des classes . . . . .	13
3.1	Séparation par l'architecture Modèle-vue-contrôleur . . . . .	17
4.1	Copie d'écran : page de connexion . . . . .	20
4.2	Copie d'écran : page de connexion . . . . .	21
A.1	Extrait des interfaces html . . . . .	i
A.2	Extrait des interfaces bases de données . . . . .	ii
A.3	Interface administrateur de gestion des articles . . . . .	iii
A.4	Interface administrateur de gestion des utilisateurs . . . . .	iii
A.5	Interface administrateur de gestion des transactions . . . . .	iii
A.6	Interface distributeur de gestion des ventes . . . . .	iv
A.7	Interface distributeur de creation d'utilisateurs . . . . .	iv
A.8	Interface distributeur de creation de billets . . . . .	iv
A.9	Interface de la caisse . . . . .	v

# Glossaire

**AJAX** est un acronyme signifiant Asynchronous JavaScript and XML (« XML et Javascript asynchrones »), et désignant une solution informatique libre pour le développement d'applications Web.

**Apache HTTP Server** souvent appelé Apache est un logiciel de serveur HTTP produit par l'Apache Software Foundation. C'est le serveur HTTP le plus populaire du Web.

**API** (Application Programming Interface ou API) est un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques par une bibliothèque logicielle, un système d'exploitation ou un service.

**CSS** (Cascading Style Sheets : feuilles de style en cascade) est un langage informatique qui sert à décrire la présentation des documents HTML et XML.

**Diagramme FAST** (Function Analysis System Technic) correspond à une méthode bien adaptée à l'étude de réalisation technologique existante. Lorsque les fonctions sont identifiées, elle les ordonne logiquement pour aboutir aux solutions techniques de réalisation.

**Javascript** est un langage de programmation de scripts principalement utilisé dans les pages web interactives. C'est un langage orienté objets à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes

**MySQL** est un système de gestion de base de données ( voir SGDB) relationnelle.

**Open source** La désignation Open Source (code source libre en français) s'applique aux logiciels dont la licence respecte des critères précisément établis par l'Open Source Initiative, c'est-à-dire la possibilité de libre redistribution, d'accès au code source, et de travaux dérivés.

**PDF** (Portable Document Format) est un langage de description de pages créé par Adobe Systems. La spécificité du PDF est de préserver la mise en forme (polices d'écritures, images, objets graphiques etc)

**PHP** (Pre Hypertext Processor), est un langage de script principalement utilisé pour produire des pages web dynamiquement.

**Serveur Web** voir serveur HTTP. Serveur Web désigne également l'ordinateur (de manière physique) sur lequel fonctionne un serveur HTTP (logiciel).

**SGBD** (système de gestion de base de données) est un ensemble de programmes qui permet la gestion et l'accès à une base données.

**SQL** (Structured query language), ou langage structuré de requêtes, est un pseudo-langage informatique (de type requête) standard et normalisé, destiné à interroger ou à manipuler une base de données relationnelle.

**UML** (Unified Modeling Language, « langage de modélisation unifié ») est un langage graphique de modélisation des données et des traitements. C'est une formalisation très aboutie et non-proprétaire de la modélisation objet utilisée en génie logiciel.

**URL** abréviation de « Uniform Resource Locator ». Chaîne de caractères (codée en ASCII, donc utilisant l'alphabet anglais, sans accent) utilisée pour adresser les ressources dans le World Wide Web. Elle est aussi appelée « adresse Web ». URL est un standard de l'IETF.

# Introduction

## Contexte

Lorsqu'un organisme désire organiser un événement (à accès restreint), il fait appel à une entreprise de billetterie telle que France billet ou la FNAC et redistribue ces billets dans différents points de vente. Lorsqu'il propose des services annexes tels que la vente de cafés, il arrive que celui-ci fasse crédit au client, ce qui implique une gestion des dettes de chaque client souvent traduite par un carnet où sont notées chaque dette.

## Besoins

Certains organismes tels que les associations à but non lucratif, ne voulant pas générer des frais supplémentaires, voudraient toutefois ne pas dépendre des entreprises de billetterie auxquels ils devraient reverser une part non négligeable de la valeur du billet ; de plus la redistribution des billets sous-entend un approvisionnement régulier des billets physiques dans chaque point de vente impliquant une gestion des stocks. La gestion d'un carnet devient au bout d'un certain temps, ou au bout d'un certain nombre de clients, compliquée ; elle peut être remplacée par un système de billet à crédit.

---

## Objectifs

Pour toutes ces raisons le besoin d'un système fiable, **efficace** et **gratuit** de gestion de billets électroniques se fait ressentir.

C'est à dire un système qui permet :

- de **délivrer des billets électroniques** au lieu de faire une tournée de distribution de billets physiques à chaque revendeurs.
- de **gérer les billets** déjà vendus (à qui, par qui, et quand).
- de **pouvoir assurer le retour des billets** via une interface facilité par code barre par exemple
- de **memoriser un crédit où dette** pour chaque clients.

Ce système doit aussi être :

- **sécurisé** afin d'assurer les services demandés
- **portable** afin d'être facilement deployable par des non spécialiste et toucher un large public
- **peu onereux** vu que le public est les association à but non lucratifs

Nous avons décidé d'appelé le produit fini LightPaw (Patte blanche) en référence à la fable de La Fontaine «Le loup, la chèvre et le chevreau» ou l'expression «montrer patte blanche» est utilisé pour dire «Donner un signe de reconnaissance pour être autorisé à entrer dans un lieu» tout comme les billets.

# Chapitre 1

## Modélisation du domaine

Ce chapitre a pour but de modéliser l'attente des associations, en se basant sur la situation actuelle du système. Et enfin proposer un modèle de fonctionnement.

### 1.1 Définitions des acteurs

#### 1.1.1 Les acteurs en présence

Dans le processus d'organisation d'un évènement liée à une vente (de bien ou de service) on distingue cinq type d'acteur composant une association :

**Les membres organisateurs :** Sont les acteurs qui administrent les rôles des autres acteurs et qui fournissent le bien ou le service à vendre.

**Les membres actifs :** sont les acteurs qui apportent leurs aides aux organisateurs en réalisant les ventes et en fournissant les biens ou services attendus.

**Les partenaires extérieurs :** sont des acteurs qui réalisent les ventes et qui peuvent être habilité a fournir certains biens ou services.

**Les membres passifs :** sont les acteurs proche des autres membres, qui sont demandeurs des biens ou des services proposés par les organisateurs

**Les clients externes :** sont des acteurs inconnu des autres membres, ayant les mêmes attentes que les membres passifs.

#### 1.1.2 Les acteurs modélisés

Le système d'acteurs réels, présentant des redondances du point de vue de l'organisation de la vente, nous avons donc choisi de simplifier la modélisation des acteurs.

En effet il s'avère que les partenaires extérieurs et les membres actifs effectuent les mêmes actions et ainsi hérite des même responsabilité, on peut donc factoriser ces deux acteurs en une seul classe independemment de la confiance qu'on leurs accorde.

De même les membres passifs et les clients ont les mêmes attentes. La connaissance de l'identité des clients externes pouvant être compensée par une identification préalable, nous

avons factorisé ces deux acteurs en une seule classe.

On modélise alors les acteurs sous les dénominations et regroupements suivants :

**Client** : personne physique bénéficiaire du service ou du bien.

**Opérateur** : personne effectuant l'acte de vente et la vérification des billets auprès du client.

**Administrateur** : personne qui administre les acteurs et les articles mis à la disposition des clients.

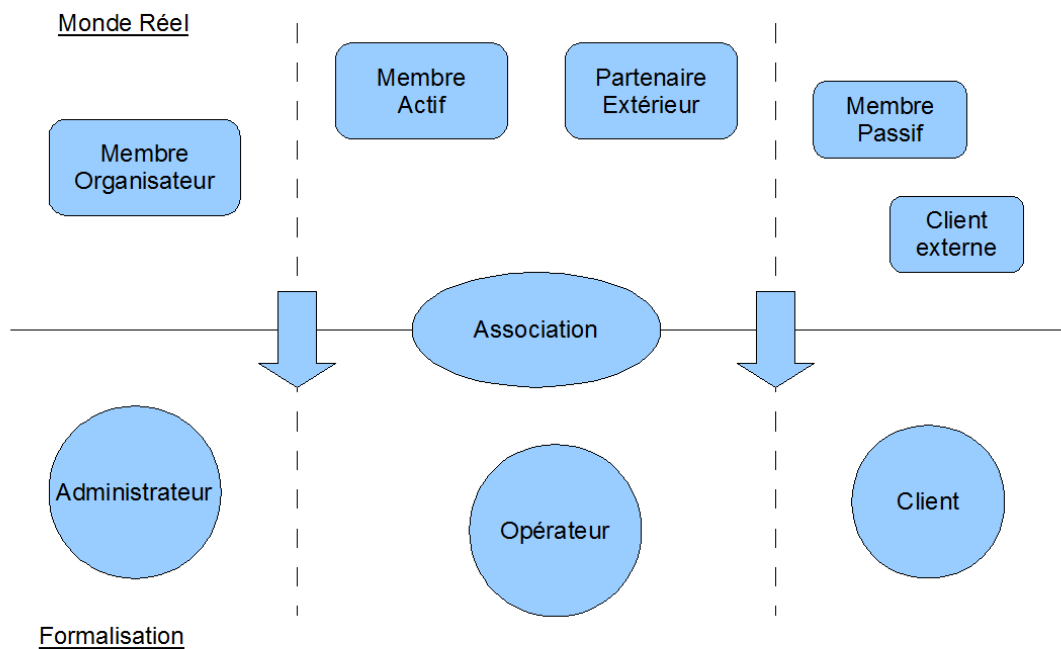


FIG. 1.1 – Schémas des acteurs d'une organisation

## 1.2 Définitions des interactions

Nous nous intéressons aux transactions de vente entre les différents acteurs.

### 1.2.1 Les ventes traditionnelles

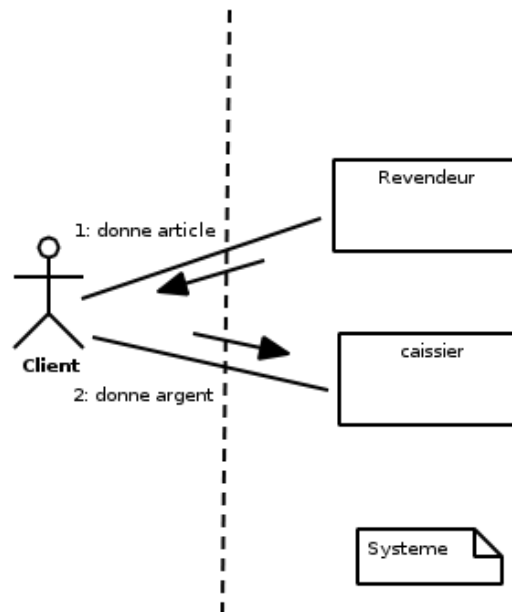


FIG. 1.2 – Diagramme UML de communication représentant une vente traditionnelle

Traditionnellement une vente peut se résumer à un flux de bien ou de service du vendeur au client suivi d'un flux monétaire du client vers le caissier.

Le vendeur et le caissier appartenant au même système cette transaction en 2 étapes peut être synchrone ou asynchrone (différé dans le temps).

### 1.2.2 Le modèle de vente

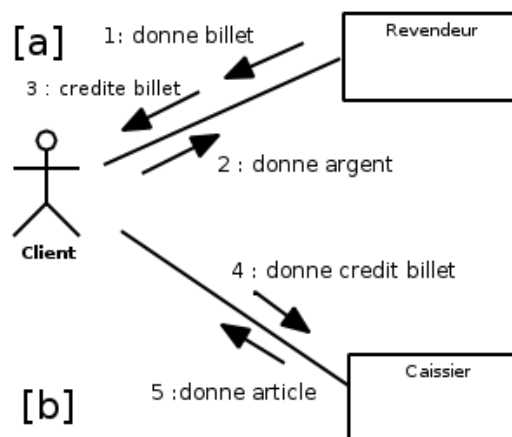


FIG. 1.3 – Diagramme UML de communication représentant le modèle de vente

Dans ce nouveau système, l'échange entre le revendeur et le client se fait en deux étapes :

- Le revendeur fournit un billet au client
- Le client crédite son billet, en reversant une somme d'argent au revendeur [a]

Puis le client obtient du caissier un article, en échange d'une diminution du crédit de son billet. [b]

La vente est donc nécessairement asynchrone du fait de l'introduction du billet et génère ainsi deux transactions atomiques faisant l'objet d'un échange d'article ([a] : rechargement de billet et [b] : bien ou service attendu).

### 1.3 Définition des rôles

Le modèles des acteurs et celui des transactions précédemment définis nous amène à définir les rôles des acteurs comme suit :

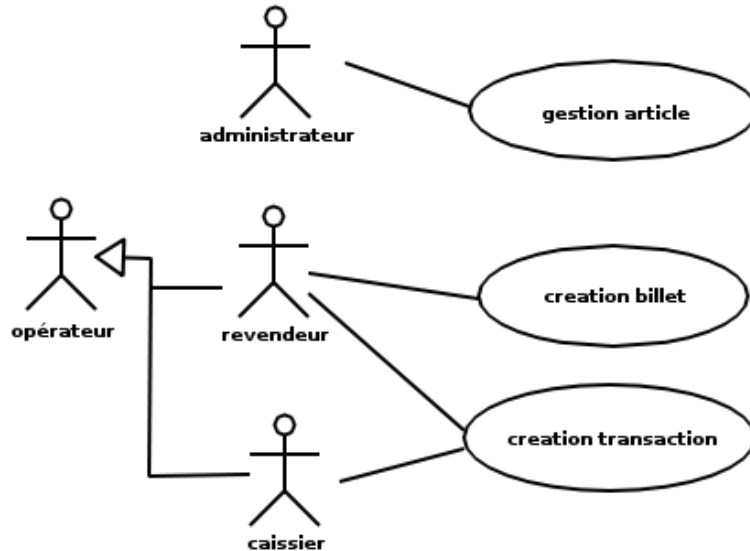


FIG. 1.4 – Diagramme UML de cas d'utilisation : les rôles des acteurs

#### 1.3.1 Administrateur

Le premier rôle de l'administrateur est la gestion des utilisateurs du système. D'autres fonctions lui sont associées, telles que le contrôle des Billets, Transactions. De plus nous avons choisit d'attribuer la gestion exclusive et intégrale des articles aux administrateurs, seuls bénéficiaires d'une confiance maximale. En effet chez les opérateurs ce droit pourrait poser des problèmes de contrôle sur les services vendus.

#### 1.3.2 Opérateur Vendeur

Selon le modèle les opérateurs dans leur fonction de «vendeur» doivent avoir la possibilité de créer des billets ; la suppression d'un billet restant à la discrétion de son propriétaire. Ils peuvent en outre générer certaines transactions de type service telles que le rechargement.

#### 1.3.3 Opérateur Caissier

En accord avec le modèle les opérateurs dans leur fonction de caissier peuvent générer des transactions de type B.

# Chapitre 2

## Conception

Ce chapitre a pour but de d'expliquer et de décrire nos choix de la manière dont nous avons implanter le système.

### 2.1 Analyse Fonctionnelle

Le modèle présenté précédemment repose principalement sur la réalisation de la fonction de service "Effectuer une transaction". Cependant d'autres préoccupations liées à la réalisation d'objectifs nous amènent à considérer ces autres fonctions de service :

#### 2.1.1 Contrôle des Transactions

Afin de détecter au plus tôt des possibles irrégularités ou tentatives de fraude il nous est paru nécessaire de fournir un système de contrôle des transactions effectuées.

#### 2.1.2 Authentification des Utilisateurs

Dans le contexte associatif il est courant que plusieurs individus aient à assumer temporairement ou non les mêmes fonctions voir d'en changer. Ainsi certains anciens membres pourraient utiliser les informations acquises ultérieurement pour outrepasser leurs droits . Il nous est donc paru nécessaire, en vue de respecter l'objectif sécuritaire, d'identifier et authentifier chaque intervenants d'un processus du système.

Nous avons donc décider d'ajouter au système une entité utilisateur déclinée par statut, ainsi que les outils de gestion des utilisateurs et d'authentification associés.

Cette fonctionnalité permet de plus d'assurer, une responsabilisation des acteurs, une traçabilité des opérations et offre de perspectives intéressantes de fidélisation des clients.

Une analyse fonctionnelle plus poussée utilisant l'outil FAST (Functional Analysis System Technique) de ces services nous a permis de mettre en évidence les éléments techniques ( cf. diagramme Fast) nécessaires au fonctionnement du système.

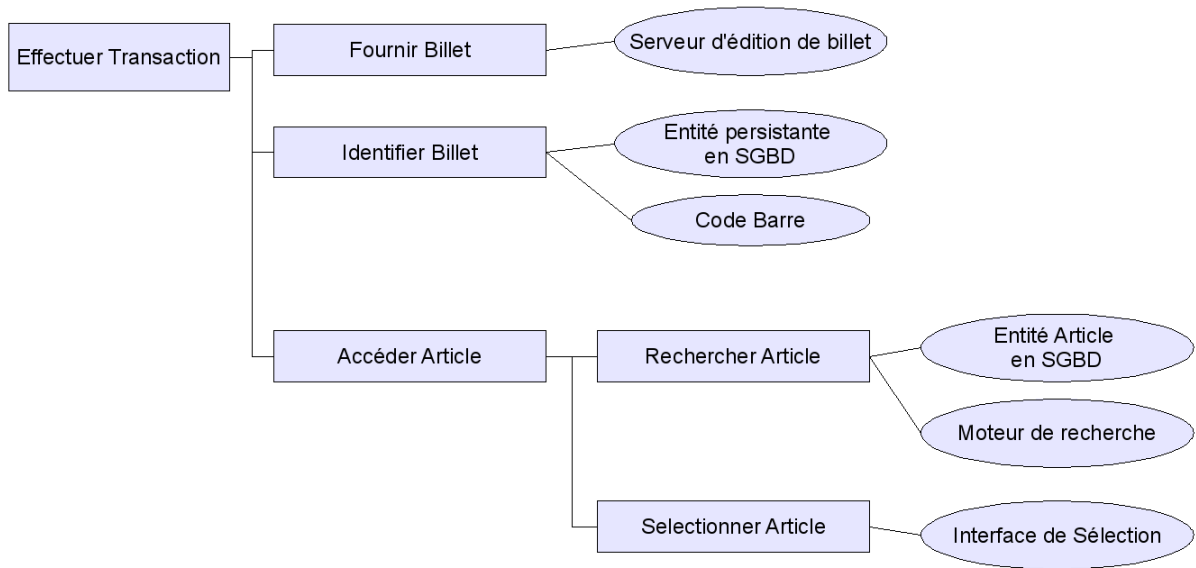


FIG. 2.1 – Diagramme de la fonctionnalité Réalisation des transactions  
*Le modèle de vente asynchrone permet le partage de la transaction A entre plusieurs ventes différentes. Ce modèle impose ainsi la persistance des billets.*

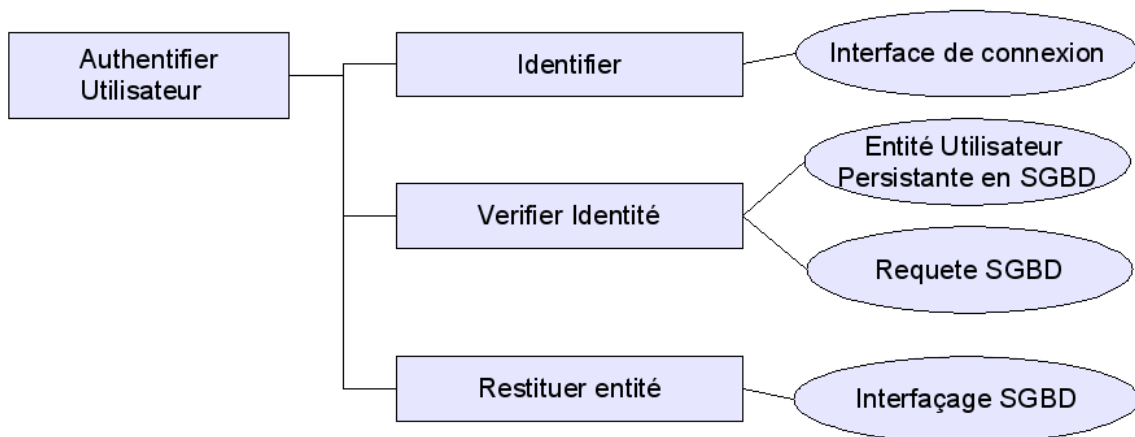


FIG. 2.2 – Diagramme de la fonctionnalité Authentification des utilisateurs  
*Pour assurer l'authentification des billets la persistance des données utilisateurs est nécessaire. De plus la persistance des données utilisateurs permet une plus large perspective d'évolution.*

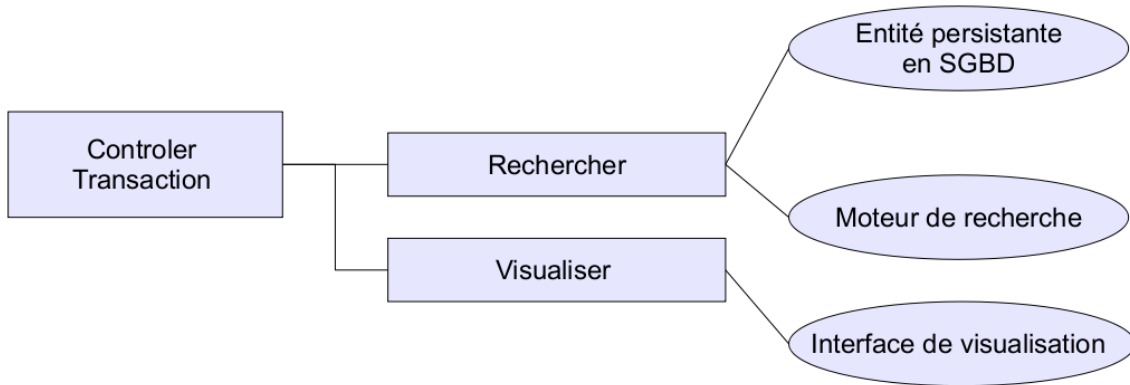


FIG. 2.3 – Diagramme de la fonctionnalité Contrôle des transactions

*A la lumière de cette analyse il apparaît donc que les entités Utilisateur (regroupant Administrateur, Opérateur) Billet, et Article doivent être persistantes et bénéficier d'un système de stockage, de gestion dédié, d'un moteur de recherche et d'interface de visualisation ergonomique. De plus l'entité Billet au vu de ces manipulations fréquentes ne serait ce par l'utilisateur, nécessite la mise en place d'un système d'édition physique ainsi qu'un système d'identification et de lecture rapide.*

## 2.2 Choix Architecturaux

L'un des objectifs premier du système LightPaw est de pouvoir fonctionner à travers un réseau dispersés géographiquement. Nous avons donc préférentiellement opté pour un modèle client-serveur.

Par commodité et souci d'accessibilité nous avons alors opter pour une infrastructure Web-Service délégrant ainsi la majeure partie de l'application cliente au navigateur Web.

Ainsi le système se résume à des applications fonctionnant sur un ou plusieurs serveurs dialoguant avec les acteurs par l'intermédiaire d'interface HTML, et accédant aux données par l'intermédiaire d'un système stockage.

### 2.2.1 Architecture du serveur

Nous avons choisi d'organiser notre système selon l'architecture MVC : Modèle vue contrôleur

#### Le patron MVC : Modèle vue contrôleur

Pour pouvoir séparer logiquement les différentes parties du système de LightPaw (afin de permettre une évolution plus aisée) nous avons décidé d'utiliser le patron Modèle-Vue-Contrôleur Celui ci sépare les moteurs qui manipulent les données, les scripts d'affichage et de la partie de contrôle (cf diagramme 2.4).

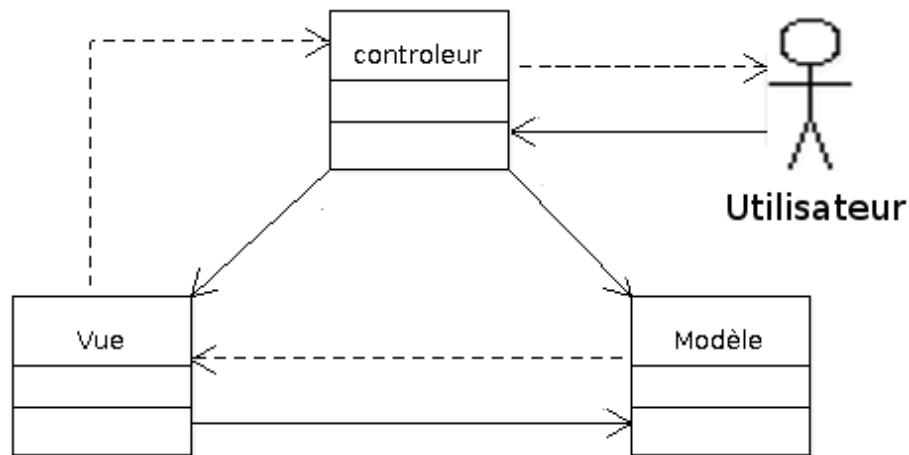


FIG. 2.4 – Diagramme UML de communication du patron Modèle-vue-contrôleur

De plus nous avons opter pour une séparation des applicatifs liée aux objets manipulés. Ainsi on peut distinguer 4 applications que nous appellerons moteurs liées aux billets, articles, transaction, utilisateurs pouvant chacun réaliser les actions atomiques suivantes :

- Création
- Suppression
- Modification

A ces moteurs s'ajoutent un éditeur de billet délivrant les billets physiques dans un format imprimable.

### 2.2.2 Architecture des interfaces

Nous avons décidé de séparer les interfaces selon 2 niveaux, le premier distinguant les interfaces selon la fonction de leur utilisateurs, le deuxième distinguant la finalité de l'action. Ainsi on distingue 3 types d'interfaces (voir schémas 2.5), dédiées respectivement aux Administrateur, Distributeur, et Vendeur-Caissier.

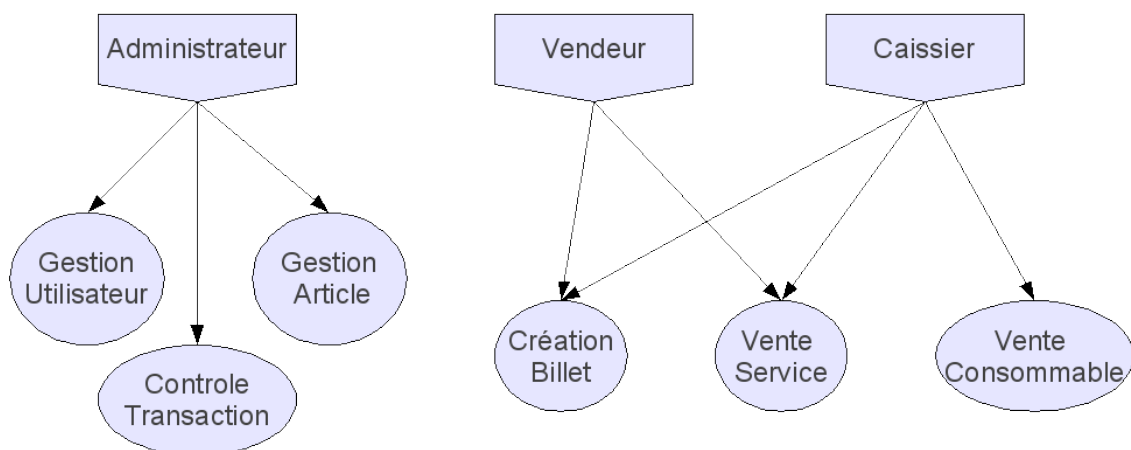


FIG. 2.5 – Interfaces

### **Interface vendeur**

L'interface vendeur est soit piloté par un distributeur physique soit intégré a un site internet qui fait office de distributeur virtuel.

Elle permet d'inscrire les nouveaux clients et de créer des billets associé à des articles et à des clients.

L'interface peut aussi être utilisée par des vendeur-caissiers.

### **Interface caisse**

L'interface de la caisse est aussi piloté par un opérateur physique soit intégré à un site internet qui fait office de vendeur-caissier virtuel.

Elle permet de valider les billets.

Le cas du vendeur-caissier virtuel s'assimile à un système d'auto validation comme il en existe dans les tramway.

### **Interface administrateur**

L'interface administrateur quant à elle ne peut être pilotée que par un administrateur physique.

Elle permet de gérer (créer, modifier ou supprimer) des acteurs, et des articles.

## **2.2.3 Architecture du système de stockage**

Afin d'assurer une homogénéité entre la structure dynamique des entités et leur structure statique dans le système de stockage ; nous avons opté pour un formalisme objet associé à des relations correspondantes dans une base de données relationnelles.

Nous distinguons ainsi 4 tables principales, nommées Transaction, Billet, Article et Utilisateur, auquel nous avons adjoint la table Client permettant de stocker des informations subsidiaire sur l'identité des clients.

## **2.3 Organisation des entités**

### **2.3.1 Utilisateur**

Afin d'assurer une gestion uniforme des utilisateurs, nous avons choisi de les distinguer par un statut définissant leur rôle : distributeur, vendeur-caissier ou administrateur (voir schémas 2.6). Chaque utilisateur est défini par un identifiant, un login, un mot de passe.

Les administrateurs et les distributeurs assumant respectivement les fonctions du même nom, à l'opposé des vendeur-caissiers pouvant assumer les diverses fonctions du groupe opérateurs (i.e vendeurs et caissiers). Il existe ainsi un niveau hiérarchique de confiance entre les 2 types d'utilisateurs distributeurs et vendeurs-caissiers.

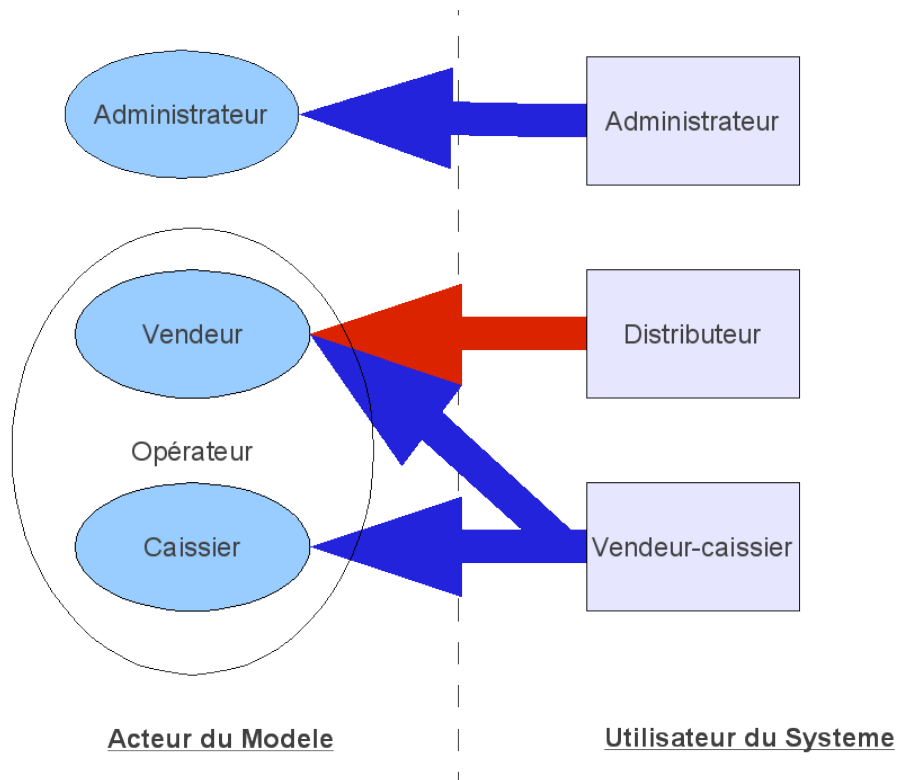


FIG. 2.6 – Utilisateurs et acteurs

### 2.3.2 Transaction

Afin d'assurer une homogénéité des données et la possibilité d'éventuelles transactions entre utilisateur, nous avons choisi de construire le modèle de transaction comme l'agrégation d'un article, et de deux billets.

### 2.3.3 Article et Billet

De manière à pouvoir restreindre l'utilisation des billets et crédits associés à des événements, produits ou opérations promotionnels, nous avons choisi de typer les articles et les Billets de manière à contrôler la cohérence des types à chaque transaction.

Le diagramme UML des classes (figure 2.7) résume l'ensemble des relations entre classes.

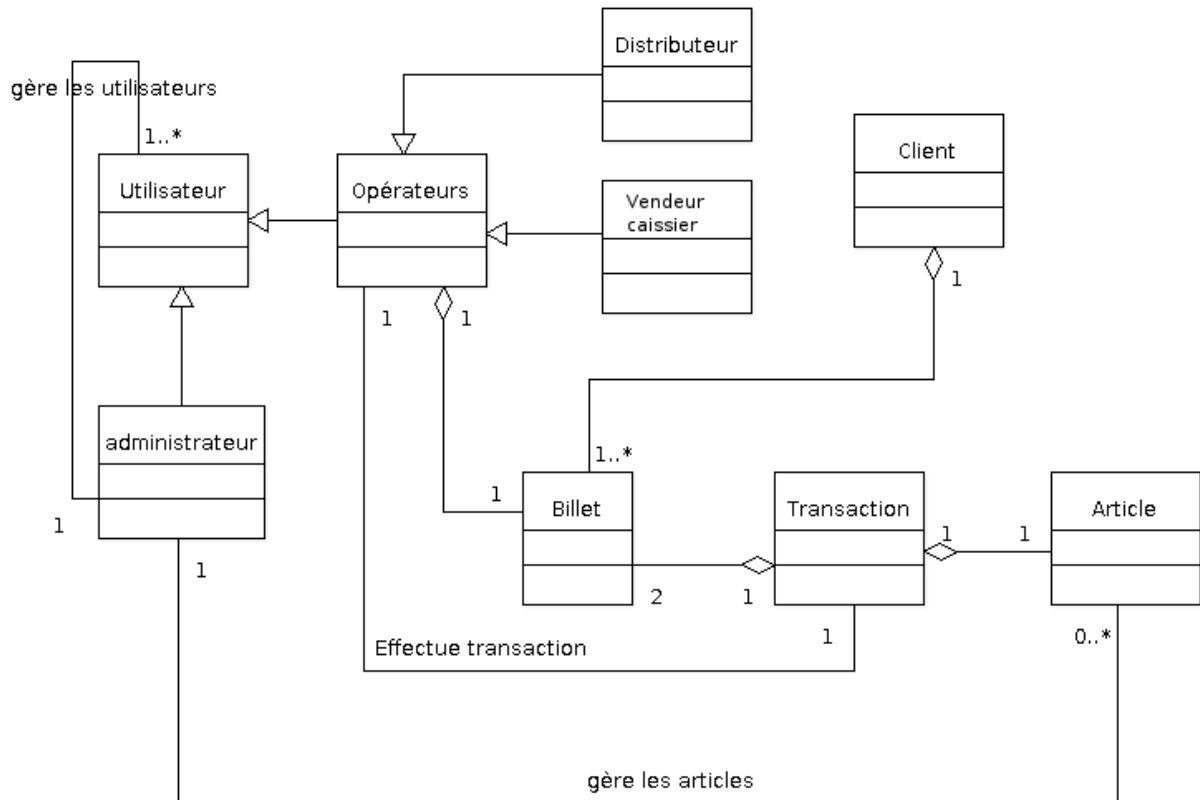


FIG. 2.7 – Diagramme UML des classes

## 2.4 Gestion de la sécurité

Un des objectifs majeur de notre système LightPaw, est de garantir la sécurité des transactions, la sureté des données et l’inviolabilité des processus métier. Pour ce faire certain point de détails ont du faire l’objet de réflexions particulières axées sur la sécurité.

### 2.4.1 Séparation des pouvoirs

La séparation native des rôles des acteurs assure au système un niveau certain de sécurité. En effet elle limite considérablement les possibilités de dérive : le vendeur ne pourras pas ainsi, par exemple créer des articles au prix désiré.

### 2.4.2 Séparation des infrastructures

La seconde idée est de séparer les infrastructures de façon physiques. De cette manière les différents acteurs ne peuvent pas pirater les infrastructure qui ne les concernent pas, étant donné qui ne savent même pas ou se trouve les autres infrastructures. Cette vision de la sécurité appuis la découpe des parties opératives. En effet elle se calque aux différents groupes de confiance présents dans le système :

**Le client** en qui on a le moins confiance, car il sont en plus grand nombre, n’as accès qu’au système de gestion des données personnelles.

**Les opérateurs** qui sont en nombre plus restreint, peuvent quant à eux acceder au système d’action sans pour autant acceder au système d’édition des billets.

L'édition des billets est toujours contrôlé par le système d'action, limitant ainsi les possibilités de fraude.

### **2.4.3 Archivage de toutes les transactions**

Pour parer à toutes tentative, ou envie de non respect aux règles du système, toutes les transactions sont archivées avec l'identifiant de l'opérateur.

De cette manière, les déviations, bien que non empêchées, peuvent être repérées et une décision peut alors être prise.

### **2.4.4 Protection des scripts**

Enfin la dernière idée est d'empêcher l'accès direct aux différents scripts et de contrôler leurs exécutions.

De cette manière le nombre de failles diminue considérablement puisque l'accès aux scripts ne se fait plus que par un seul point d'entrée.

# Chapitre 3

## Réalisation

Dans ce chapitre nous allons étayer nos différents choix quant à la réalisation de Light-Paw.

Nous avons choisi d'utiliser un ensemble de technologies qui permette la plus grande portabilité au sein du public visé.

Le public visé étant les associations à but non lucratif, il va de soit qu'il s'orientera vers des solutions à faible coût voire à coût nul ; notre système doit donc pouvoir fonctionner sur un système se basant sur les moyens donnés par les serveurs dit gratuits.

Notre choix s'est donc porté vers le PHP couplé avec le système de gestion de base de donnée MySQL présents tout deux sur la plupart des hébergeurs gratuits.

### 3.1 Réalisations relatives aux bases de données

Les tables de base de données sont calquée sur la conception du système détaillée dans le chapitre 2.

#### 3.1.1 Gestion des accès concurrentiels

Dans le cas où deux acteurs envoient une requêtes au serveurs au même instant, il faut veiller à préserver une cohérence des données. Par exemple il ne faut pas attribuer deux fois le même billet à deux personnes différentes.

#### Choix d'un moteur

La gestion des accès concurrentiels d'une base de donnée est faites par le SGBD (dans notre cas MySQL).

Cependant tous les moteurs dédiés à MySQL ne gère pas les accès concurrentiels Le moteur par défaut MyIsam gère les accès en écriture en verrouillant la table entière et ne gère pas les transactions, c'est à dire qu'en cas de problème MyIsam ne peux pas réparer les erreurs commises.

Cela poserait de sérieux problème :

- Verrouillage de la base de donnée pour un temps donné, si une opération échoue pendant une écriture, impliquant un blocage de tous les acteurs du système et une opération non effectuée entraînant une incohérence de la base de donnée.
- La perte de donnée, ou une incohérence de la base de donnée, puisqu'il n'existe pas la notions de transaction.

Pour ces raisons nous nous sommes tourné vers InnoDB, un des 2 moteurs avec BerkeleyDB, gérant les transactions. Le choix de innoDB est appuyé par sa plus grande disponibilité au sein des hébergeurs.

### Les sessionn MySQL

Une session MySQL est un lien créé lors de la connexion à la base et détruit lors de la déconnexion. Ce lien permet d'identifier l'émetteur de chaque requêtes pour ainsi gérer les transactions du point de vue de la base de donnée. Sachant que sans ce lien la base de donnée est complètement indépendante des exécutions des scripts PHP, qui peuvent alors provoquer des incohérence dans celle ci, par des accès concurrentiels.

### Mise en place au niveau PHP

Les tables présentent en base de donnée ont souvent une colonne ayant l'option «AUTO\_INCREMENT» ce qui nous évite la gestion des doublons. Cependant comme nous ne gérons pas les valeurs insérées dans ces colonnes il nous faut trouver un moyen de la récupérer.

La première idée était de refaire une requête sur la table afin de retrouver l'élément inséré, seulement ce n'est pas toujours possible car il faudrait que la partie du tuple inséré privé de la colonne ayant l'option «AUTO\_INCREMENT» soit unique (ce qui n'est pas toujours le cas).

Une seconde idée est de retrouver le tuple dont la colonne ayant l'option «AUTO\_INCREMENT» contient la plus grande valeur. Cependant cette méthode aussi n'est pas acceptable, car un tuple peut être inséré par une autre personne entre notre insertion et notre recherche de la valeur, ce qui fausse les résultats et amène à des incohérence de la base de donnée.

Une recherche parmi les fonctions de contrôle de MySql par PHP, nous a permis de résoudre le problème des colonnes à option «AUTO\_INCREMENT». Il faut pour retrouver la valeur de la colonne utiliser la fonction *mysql\_insert\_id* qui retourne le dernier identifiant généré par un champ de type AUTO\_INCREMENT, sur la connexion MySQL courante.

#### 3.1.2 Sécurisation des requêtes base de données

Afin de sécuriser les bases de données, il fallu nous prémunir contre l'injection de code dans celles ci. Pour cela nous avons utilisé la fonction PHP *mysql\_real\_escape\_string* pour préparer, les chaînes de caractère saisis par l'utilisateur, avant de les insérer en base.

Cette fonction échappe l'ensemble des caractère spéciaux.

## 3.2 Réalisations relatives à l'application serveur

### La mise en place de l'architecture Modèle-vue-contrôleur

Nous avons séparé les différentes parties dans des dossiers distincts (cf. figure 3.1)

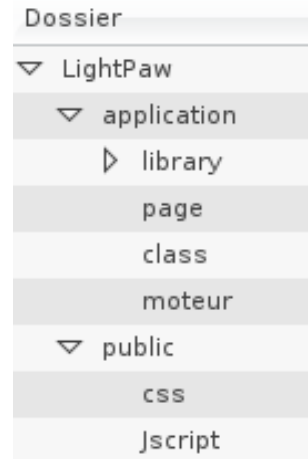


FIG. 3.1 – Séparation par l'architecture Modèle-vue-contrôleur

L'ensemble des fichiers est séparé en deux parties :

- les fichiers publics dans le dossier «public»
- les fichiers de l'application (privés) dans un dossier «application».

**Les scripts de vues** sont regroupés dans le dossier «page».

**Les moteurs** sont regroupés dans le dossier «moteur».

**Les classes (modèles)** sont regroupées dans le dossier classe.

**Les librairies**, classes et fonctions, qui sont réutilisables pour un autre projet sont regroupées dans le dossier «library».

Le tout est contrôlé par un script placé à la racine nommé «index.php».

#### Une centralisation des requêtes : «Front controller»

L'architecture MVC, utilise le concept de «Front controller». Ce script consiste à mettre en place un point de passage obligatoire pour accéder aux autres scripts PHP, celui-ci contrôle l'identité des utilisateurs et met en place l'environnement adéquat à la bonne utilisation du système.

Concrètement l'ensemble des scripts PHP sont regroupés dans un dossier dont on interdit l'accès par un fichier «.htaccess» (fichier de configuration d'apache server).

Le «front controller» faisant partie du système a les droits suffisants pour accéder aux scripts qu'il affiche après vérification de l'identité préalable de l'utilisateur.

Le «front controller» permet aussi de mettre en place l'environnement adéquat en chargeant les classes et fonctions utiles.

### Les échecs liés à la mise en place du «front controller»

Afin de contrôler totalement l'accès direct aux fichiers des scripts PHP, notre première idée était d'utiliser un système de réécriture d'URL, de cette manière l'utilisateur n'as aucune idée de la structure de stockage.

Apache server propose un module «*mod\_rewrite*» qui permet de réécrire les URL, cependant c'est un module qui est très rarement activé ou installé sur les serveurs gratuits.

Une astuce consistait à rediriger les utilisateurs lorsqu'ils entraient une URL non existante vers notre front controller qui faisait alors son travail selon la faute commise. Malheureusement, les variables envoyés par les différents scripts aux moment des redirections étaient effacées.

Nous avons donc du renoncer à la réécriture d'URL au profit d'un système de front controller à paramètre, c'est a dire que le front controller s'attend toujours à certains paramètres fournit cette fois ci par l'utilisateur dans l'URL demandé.

### 3.2.1 Les outils utiles à l'application

#### la classe champs

La classe champs est un outils qui permet de stocker des attributs tout en conservant des méta-données. Elle facilite la présentation HTML, l'édition de formulaire, la récupération des donnée d'un formulaire, et le contrôle des donnée

#### La classe base de données

Afin de permettre une meilleur portabilité de LightPaw nous avons développé un classe base de donnée qui s'occupe d'envoyer les requêtes, ainsi qu'une classe approprié aux base de donnée MySQL. Cette dernière peut être échangée par une autre classe si l'utilisateur préfère utiliser un autre SGBD.

Cette classe est agrégée par toute les entités persistantes de manière à permettre un échange transparent avec les bases de données.

Le format de requêtes que nous avons retenue est le SQL, étant donné sa généralisation dans le domaine des bases de données relationnels.

#### Les interfaces

Une série de méthode ont été développées de manière à factoriser les échanges d'informations entre le navigateur et les bases de données.

**L'interface «HTML»** Au niveau objet, des méthodes génériques tirant parties de la classe champs ont permis de générer des présentations statiques et sous forme de tables avec le navigateur. (cf annexe A.1)

**Les interfaces bases de données** Des série de méthode ont été développées pour permettre un échange transparent des informations avec les bases de données (cf annexe A.2).

De plus des méthodes statiques ont été développées permettant des fonctionnalités telles que les moteurs de recherches.

Les choix de réalisation vous ont été présentés, le chapitre suivant vous présenteras enfin le produit exploitable.

# Chapitre 4

## Résultats et prolongement

Dans ce chapitre nous présentons LightPaw, son fonctionnement selon le rôle de l'utilisateur. Puis nous passerons en revue les système similaires déjà existants, et enfin nous émettrons des possibilités d'évolutions du produit.

### 4.1 Présentation du produit

Le système vue de l'extérieur présente sept interfaces, partagées avec les 3 types d'utilisateurs, qui demandent une authentification de l'opérateur avant utilisation (voir copie d'écran 4.1).

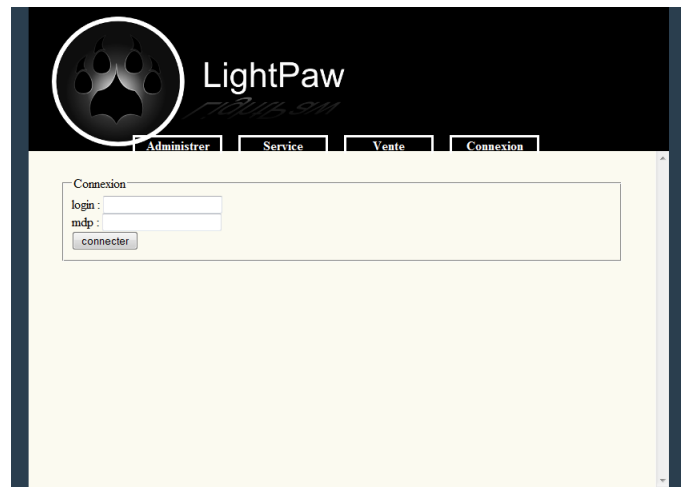


FIG. 4.1 – Copie d'écran : page de connexion

En cas de saisi de mauvais couple nom d'utilisateur/mot de passe LightPaw le signale.

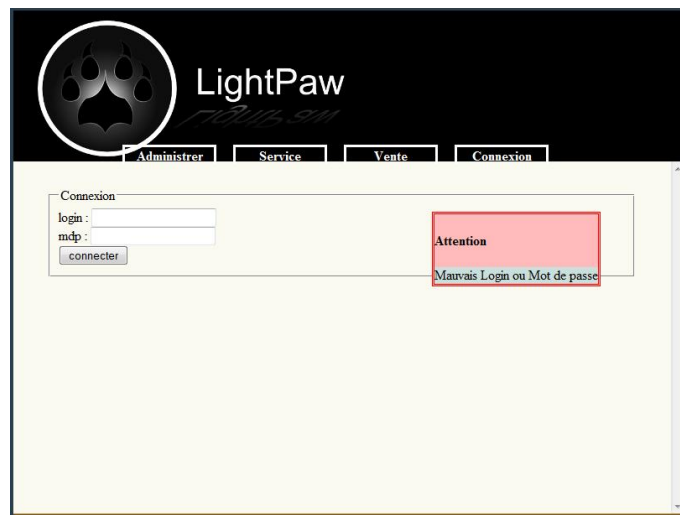


FIG. 4.2 – Copie d'écran : page de connexion

Selon le type de l'utilisateur, il a accès a différentes interfaces (voir Figure en annexe)

## 4.2 Prolongement

### 4.2.1 L'existant

- L'IATA (International Air Transport Association) a déjà développé son système de gestion de billets électroniques, mais il n'est pas accessible au grand public.
- Des élèves de centrale Lille travaillent sur un sujet similaire : «un système de titres de transports électroniques».  
Site internet du projet : <http://eleves.ec-lille.fr/~tem06/>
- La SNCF teste un système de billet commandé par internet et envoyé par MMS.  
Site internet : <http://www.altivis.fr/La-SNCF-teste-Tikefone-le-billet.html>

### 4.2.2 Les ajouts futurs

Ayant développé LightPaw avec une approche objet, il est assez simple de rajouter ou de modifier le code déjà produit, parmi les idées de perfectionnement il y a :

- Le rajout d'une nouvelle méthode d'envoi des billets par exemple par MMS
- Le rajout de classe SGBD spécifique pour garantir une plus grande portabilité
- Le rajout de script AJAX afin d'augmenter l'ergonomie.

# Conclusion

LightPaw est un logiciel open source, explorant un domaine quelque peu délaissé. D'une constitution fiable et basé sur des technologies éprouvées, cette ébauche permet déjà de répondre aux objectifs fixés et promet de nombreuses améliorations.

# Bibliographie

**php.net** référence du langage PHP

# Annexe A

## Annexes

```
1 function from_Form($list)
2 {
3     foreach($list as $ind)
4     {
5         $this->$ind->from_Form();
6     }
7 }
8
9 function to_Form($list)
10 {
11     $ret='';
12     foreach($list as $ind)
13     {
14         $ret.= $this->$ind->to_Form().'  
';
15     }
16     return $ret;
17 }
18
19 function to_Html($list)
20 {
21     $ret='<table>';
22     foreach($list as $ind)
23     {
24         $ret.= '<tr>'.$this->$ind->to_Html().'</tr>';
25     }
26     $ret.='</table>';
27     return $ret;
28 }
```

FIG. A.1 – Extrait des interfaces html

```

1 function _inserer_base($list)
2 {
3     $requete = 'INSERT INTO '.self::table.' (';
4     $part2 = 'VALUES (';
5     foreach($list as $ind)
6     {
7         $requete .= $this->$ind->nom.',';
8         $part2 .= "'".$this->$ind->valeur."',";
9     }
10    $requete = substr_replace($requete,')',-1). substr_replace($part2,')',-1).';';
11    self::$bdd->requete(self::base,$requete);
12    return $requete;
13 }
14 function supprimer_base()
15 {
16     $requete = 'DELETE FROM '.self::table.' WHERE id='.$this->id->valeur.'';
17     self::$bdd->requete(self::base,$requete);
18     return $requete;
19 }
20 function _maj_base($list)
21 {
22     $requete = 'UPDATE '.self::table.' SET ';
23
24     foreach($list as $ind)
25     {
26         $requete .= $this->$ind->nom."=".$this->$ind->valeur."',";
27     }
28     $requete = substr_replace($requete,')',-2). ' WHERE id='.$this->id->valeur.'';
29     self::$bdd->requete(self::base,$requete);
30     return $requete;
31 }
32 function maj_loc()
33 {
34     $requete = 'SELECT * FROM '.self::table.' WHERE id='.$this->id->valeur.'';
35     $tb = self::$bdd->requete($this->base,$requete);
36     $tableAssoc = $tb[0];
37     $this->remplir($tableAssoc);
38     return $requete;
39 }

```

FIG. A.2 – Extrait des interfaces bases de données

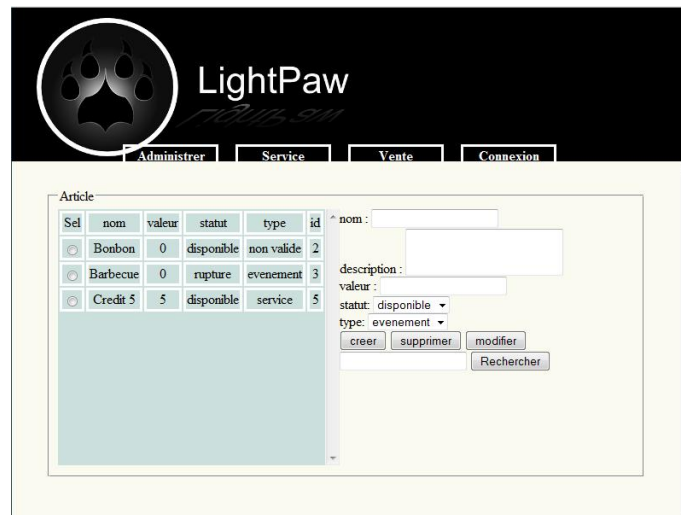


FIG. A.3 – Interface administrateur de gestion des articles

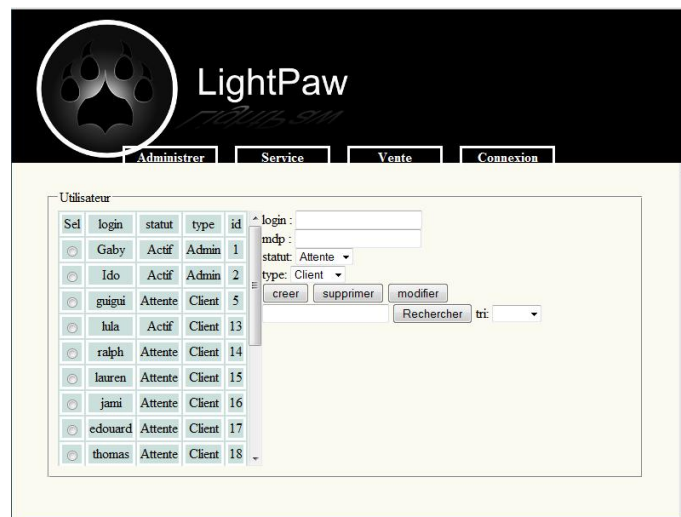


FIG. A.4 – Interface administrateur de gestion des utilisateurs



FIG. A.5 – Interface administrateur de gestion des transactions

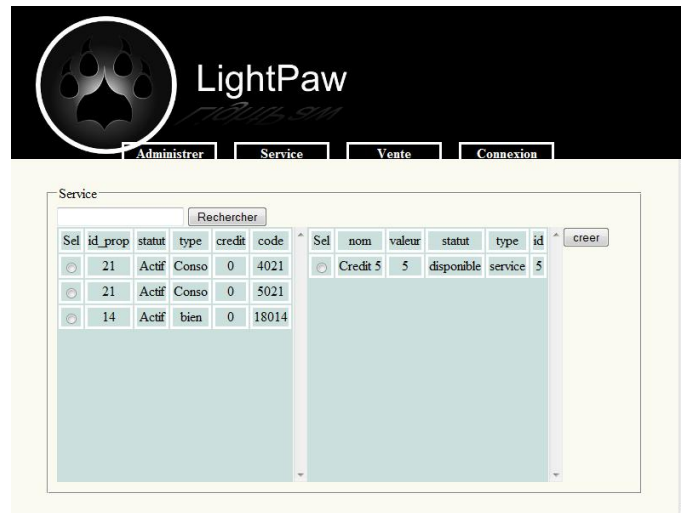


FIG. A.6 – Interface distributeur de gestion des ventes

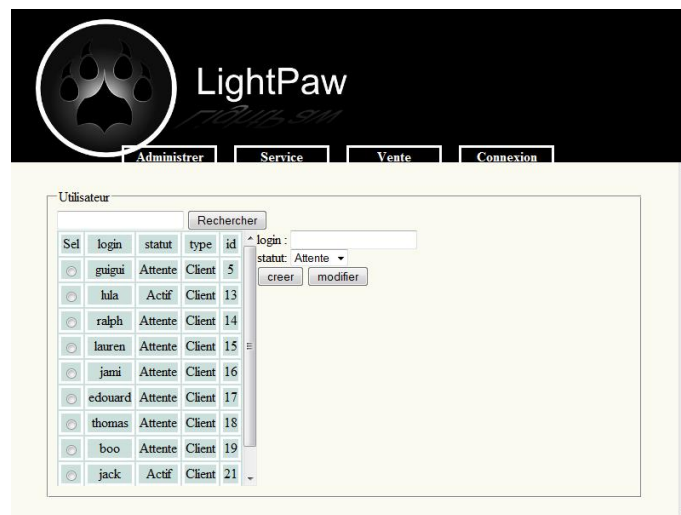


FIG. A.7 – Interface distributeur de creation d'utilisateurs

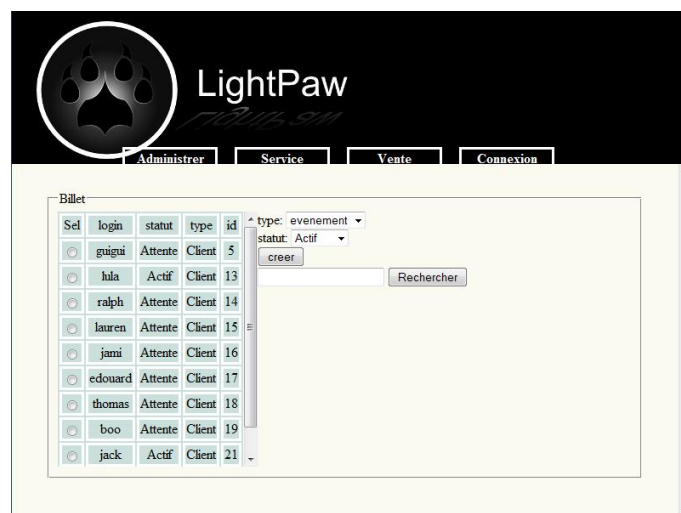


FIG. A.8 – Interface distributeur de creation de billets

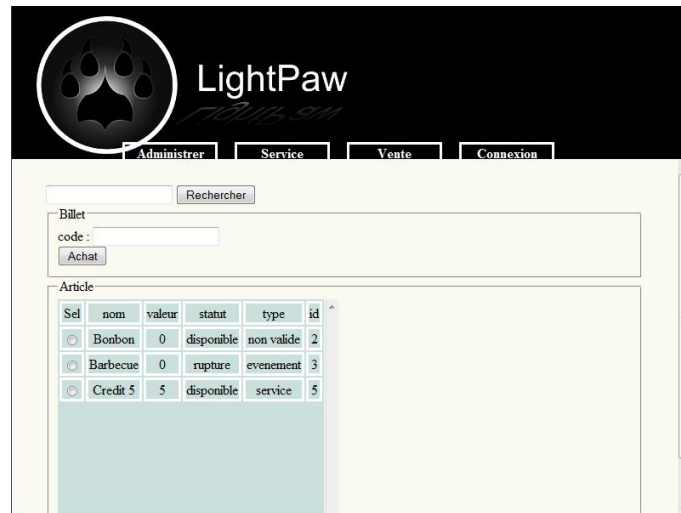


FIG. A.9 – Interface de la caisse