Département Informatique - IUT2 GRENOBLE

Année Universitaire 2006/2007
MÉMOIRE DE STAGE

GESTION DE LA FACTURATION ET DES EFFECTIFS D'UNE ÉCOLE DE MUSIQUE
DSI de la ville de Fontaine

(Stage du 02/04/2007 au 08/06/2007)

Présenté par : Romain BOSSUT
Promotion : Seconde Année

IUT : Annie CULET
IUT : Éric FONTENAS
Tuteur entreprise : Vincent KOBER

Remerciements

Je tiens à remercier en premier toute l'équipe de la DSI de Fontaine, qui m'a accueilli dans ses locaux.

Plus particulièrement, je tiens à remercier mon tuteur de stage Vincent KOBER pour la confiance qu'il m'a accordé tout au long de ce stage, et pour m'avoir laissé travaillé de manière autonome, tout en m'apportant son expertise et son recul qui m'ont permis de mener ce projet à bien. Je tiens à le remercier également pour m'avoir donné l'opportunité de suivre une journée de formation sur le framework CakePHP.

Mes remerciements vont également à Christophe ESPIAU de l'Addulact, qui s'est déplacé depuis Montpellier pour venir nous donner cette journée de formation.

Je remercie de plus Amandine FRANCE, secrétaire de la DSI, pour son professionnalisme et sa disponibilité.

Toute ma gratitude à Catherine BAUBIN et Graziella GAGLIARDO, de l'école de musique, pour leur enthousiasme vis-à-vis de ce projet et leur travail précieux de test et de retour d'information.

Enfin, j'adresse un clin d'œil amical à Yoann CHABERT, également stagiaire à la DSI, pour sa compagnie, agréable de surcroît.

Table des matières

1 Introduction

Suite à un entretien avec le directeur des systèmes d'information (DSI) de la ville de Fontaine, M. KOBER, j'ai intégré son service pour un stage de deux mois à compter du 2 Avril 2007.

Cette DSI, rattachée à la commune de Fontaine, constitue le pôle informatique principal de la ville, et dispose de locaux dans les bâtiments du CCAS de Fontaine, dans lesquels le stage s'est déroulé. Le service assure au quotidien la maintenance des postes, réseaux, et applications dans les différentes administrations et structures publiques de la commune (écoles, mairie ...). Il se charge également de la maîtrise d'ouvrage auprès de différents prestataires. La DSI est parfois amenée à conduire des projets de développement pour répondre à des besoins spécifiques, et fait régulièrement appel à des stagiaires à cet effet.

La DSI a clairement misée sur le logiciel libre pour bon nombre de ses applications, et utilise les services de l'ADDULACT[1]. Cette association offre un accès à de nombreux logiciels libres dédiés aux collectivités territoriales, des formations pour les développeurs, ainsi qu'un accès à une plate forme de développement collaboratif. Cette association favorise ainsi la mutualisation des solutions créées par les différentes collectivités.

La DSI est également en relation avec le SITPI, qui se charge d'héberger les plus grosses applications d'un groupement de commune, et offre également un espace de formation informatique dédié au personnel communal.

Mon intervention se situe au niveau de l'école municipale de musique de Fontaine, qui accueille environ 300 élèves, et dispense des cours tout au long de l'année scolaire.
Il m'a été demandé de concevoir une nouvelle version d'un outil de gestion de l'école de musique municipale, réalisé en 2006 par un stagiaire en seconde année de DUT au département informatique de l'IUT2. Le besoin principal exprimé par l'école de musique fut pouvoir gérer conjointement inscription des élèves, paiements et facturation, qui présentent des spécificités métiers dont les logiciels actuels de gestion d'école de musique ne tiennent pas compte. Le précédent logiciel répondait en partie à ces attentes, mais posait de gros problèmes d'ergonomie, de souplesse et de fiabilité, il a donc fallu prendre en compte ces nouveaux aspects dans l'analyse.

Le but de cet outil est de faciliter la vie de l'administration de l'école de musique en prenant en charge de manière automatique la création des factures et des documents comptables relatifs aux paiements des élèves, tout en laissant la possibilité à la direction d'assurer elle-même un suivi de l'état des paiements en temps réel. En gérant de plus les

[1]Association des Développeurs et des Utilisateurs de Logiciels Libres pour l'Administration et les Collectivités Territoriales

aspects spécifiques à la gestion des paiements internes à l'école de musique, le logiciel doit permettre de formaliser certains processus et certaines règles, réduisant ainsi le risque d'erreurs humaines.

La première phase de ce travail a été d'identifier et d'analyser au sein de l'école de musique le besoin, afin d'en définir les contours et les limites.

Le choix de la solution technique adoptée s'est ensuite fait en fonction des technologies disponibles et utilisées au sein de la DSI, il a donc été décidé dès le début de produire une application sous forme de site WEB et d'utiliser un environnement Apache/PHP/MySQL. Dans cet espace là, l'opportunité d'utiliser le framework CakePHP pour la réalisation de l'application s'est offerte au travers l'offre d'une journée de formation organisée par l'ADDULACT sur ce sujet. Une grande partie du travail a donc été d'expérimenter pour la première fois au sein de la DSI ce framework, afin d'en tirer parti pour ce projet en particulier, mais aussi pour amener une connaissance sur ce framework qui pourrait dans l'avenir être réutilisée de nouveau par la DSI. Les derniers aspects sur ce stage sont la phase d'intégration du nouveau logiciel dans son contexte technique et humain, et une réflexion sur les améliorations possibles.

2 Analyse de l'existant

2.1 Contexte organisationnel

2.1.1 L'école de musique

Prestation

L'école municipale de musique de Fontaine se situe au second étage de l'école primaire Anatole FRANCE de Fontaine. Elle accueille entre 200 et 300 élèves par an et de tous âges, et leur propose diverses prestations (solfège, pratique collective, cours individuels . . .) à des tarifs adaptés à leurs moyens financier.

Personnel

L'équipe pédagogique est composée d'une trentaine de professeurs qui assurent les cours au quotidien. L'administration de l'école est assurée par sa directrice, Catherine BAUBIN, assistée de Graziella GAGLIARDO, actuellement au poste de secrétaire. Catherine BAUBIN et Graziella GAGLIARDO sont toutes deux bien formées à l'utilisation des logiciels bureautiques courants. Dès le premier contact, il était évident que l'utilisation de l'outil informatique ne les rebuterait pas.

2.2 Contexte technologique

2.2.1 La DSI

La DSI dispose de nombreux postes informatiques tous en réseaux, la plupart sous Windows XP. Elle dispose également d'un serveur (nommé WebOracle) qui héberge les services WEB de la DSI, grâce à un serveur Apache 2, avec le module PHP5 et la base de données MySQL 5.

2.2.2 L'école de musique

La secrétaire et la directrice disposent chacune d'un poste informatique sous Windows XP avec la suite OpenOffice installée. Ces deux postes sont en réseaux et reliés à internet par connexion ADSL, et peuvent donc accéder à l'intranet de la DSI via une connexion sécurisée HTTPS. Afin de gérer le suivi pédagogique des élèves, elles disposent d'une base sous forme de tableur Calc. Pour ce qui est de la gestion des paiements et des factures, elles utilisent le logiciel mis en place par le précédent stagiaire avant mon arrivée.

2.3 Description de la solution existante

N'ayant pas pu avoir de contact avec l'école de musique les premiers jours de mon stage, j'ai basé une grande partie de l'analyse du besoin en identifiant les fonctionnalités que le logiciel précédent offrait. Cette première analyse a pu par la suite être complétée

par les remarques de la directrice et de la secrétaire de l'école de musique sur cette première solution.

2.3.1 Possibilités du premier logiciel

Gestion des effectifs

Le logiciel permet de saisir, par années, toutes les informations concernant les élèves de l'école, ainsi que celles concernant leurs responsables légaux.

Gestion des inscriptions

Chaque année, la secrétaire peut inscrire ou réinscrire des élèves à une prestation donnée, en indiquant le nombre de paiements (de 1 à 3).

Gestion des paiements

Des paiements de 3 natures différentes peuvent être enregistrés : chèques, espèces, et chèques jeunes. À chacun de ceux-ci est associé un numéro de reçu, que la secrétaire peut modifier elle-même afin de gérer les paiements concernant plusieurs élèves ou plusieurs échéances.

Génération des documents comptables

Pour chaque échéance trimestrielle, deux types de documents peuvent être édités : des documents comptables (bordereaux des chèques, état des paiements) sous forme de fichier Excel, et des factures, sous forme d'une page WEB à imprimer.

2.3.2 Critique des utilisateurs

Certaines erreurs ont apparemment été commises lors de la conception du précédent logiciel, ce qui à amené la DSI à refondre ce projet. Afin d'apprendre de ces erreurs, nous avons demandé à l'école de musique d'identifier pour nous les aspects à revoir.

Définition des limites du projet

D'après les témoignages de Catherine BAUBIN, et de Vincent KOBER, les limites du projet furent mal fixées, ce qui a conduit le stagiaire à faire un projet beaucoup trop long et qui n'a pu être finalisé correctement. Lors du précédent projet, la gestion de l'aspect pédagogique avait été envisagé (gestions des notes, appréciations des professeurs etc.). Ceci a freiné le développement de l'essentiel : la gestion des inscriptions et de la facturation.

Manque de souplesse et de visibilité

C'est certainement cet aspect qui posait le plus de difficultés à la secrétaire et à l'école de musique. En effet, la secrétaire pouvait facilement ajouter des inscriptions et des paiements, mais avait peu de visibilité ensuite sur les informations qu'elle venait de saisir. Ceci conduisait l'école de musique à mener toutes ces opérations "à l'aveugle",

sans aucune possibilité de modifier ou même de se rendre compte d'une erreur de saisie avant l'opération de clôture du trimestre, qui est irréversible.

Cela mettait donc une pression importante lors de l'utilisation du logiciel, toute erreur de la part de l'utilisateur pouvant se révéler critique.

Possibilité de configuration réduite

Beaucoup d'opérations liées à des paramétrages (grille de tarifs, dates des échéances, mode de paiements ...) n'étaient accessibles d'aucune manière à l'école de musique, qui devait faire appel à la DSI, à charge de celle-ci ensuite d'effectuer les modifications nécessaires directement dans la base de données.

Erreurs indécelables

Du fait du mode de calcul adopté dans le précédent logiciel, une bonne partie des informations concernant les paiements étaient perdues, ce qui rendait difficile la détection de l'origine d'une erreur. Il est arrivé en effet que lors de la génération d'un état comptable, certains calculs se soient révélés faux sans que l'on puisse en connaître la cause (Erreur de saisie ? À quel moment ?). De ce fait la secrétaire a du vérifier manuellement une bonne partie des paiements, ce qui rendait caduque le gain de productivité apporté par le logiciel.

Points positifs de l'ancienne solution

Ces quelques défauts ne doivent pas éclipser le bon travail du précédent stagiaire sur la partie modélisation et analyse du besoin. En effet le précédent stagiaire à réalisé son stage en grande partie dans les locaux de l'école de musique, ce qui lui a permis de bien aborder la problématique. De fait, les fonctionnalités prévues par le premier projet furent au final bien celles attendues par l'école de musique, c'est plus leur mise en œuvre et l'aspect ergonomique qui a été négligé.

2.3.3 Conséquences sur le projet

L'analyse du besoin premier ayant été plutôt bien menée sur le premier projet sur les aspects facturations et inscription, celle de ce nouveau logiciel la reprendra dans les grandes lignes, en éliminant tous les aspects pédagogiques qui ne correspondent pas à un besoin immédiat de l'école de musique, afin de dégager du temps pour améliorer la qualité logicielle. L'amélioration de la qualité concernera donc essentiellement l'ergonomie, la fiabilité, et la maintenabilité du projet.

2.4 Analyse du besoin

2.4.1 Description textuelle du besoin

Inscriptions

Avant le début de l'année, l'école de musique envoie aux élèves déjà inscrits une fiche de réinscription, qui est également disponible aux nouveaux arrivants. Cette fiche est ensuite retournée à l'école de musique avant le début des cours. La secrétaire saisit les élèves avec leurs responsables légaux, s'ils ne sont pas déjà enregistrés dans le logiciel, au fur et à mesure qu'elle reçoit les fiches d'inscriptions. Cette saisie concerne le nom et le prénom de l'élève, sa date de naissance, son école ou lycée avec sa classe, ses numéros de téléphone. Les informations concernant le responsable sont ses noms et prénoms, son adresse postale, ses numéros de téléphone, et son quotient familial, utile pour déterminer par la suite un tarif adapté.

Ensuite la secrétaire saisit l'inscription pour l'année en cours avec la prestation et la classe tarifaire (adulte ou enfant par exemple), puis indique si l'élève souhaite avoir une gestion aménagée de ses horaires, et s'il souhaite s'acquitter de sa facture en 1,2 ou 3 paiements.

Factures

Au début de chaque échéance trimestrielle, la secrétaire imprime grâce au logiciel les factures qu'elle envoie aux élèves inscrits. Le montant d'une facture correspond au tarif à l'année pour la prestation choisie par l'élève, pour une classe de tarifs donnée (adulte ou enfant par exemple), et le quotient familial de son responsable légal, le tout divisé par le nombre de versements souhaités, plus ou moins un éventuel trop perçu ou moins perçu sur la facture précédente.

Il est souhaitable que la facture présente le détail des paiements précédents et fasse apparaître les éventuels reports de solde ou impayés.

Paiements

À chaque réception d'un paiement, la secrétaire l'enregistre dans le logiciel pour l'échéance en cours, avec sa date, son mode de paiements (chèque, espèces, chèque jeune ...), et son montant, ainsi qu'un numéro de reçu du trésor public. Ce numéro est unique pour chaque versement reçu par l'école de musique. Si un versement concerne par exemple deux élèves, alors le même numéro de trésor public sera affecté à chacun des deux paiements.

Échéances

Avant la clôture de l'échéance, la secrétaire prend rendez-vous avec le Trésor Public afin de faire le point sur la situation comptable. La secrétaire visualise donc les impayés, et décide en fonction de leurs montants de les reporter sur la facture suivante, ou bien de les faire prendre en charge par le Trésor Public, au cas par cas. Dans le cas d'une prise en charge par le Trésor Public, la facture de l'élève sera donc considérée comme payée pour l'école de musique, mais il devra s'en acquitter ensuite directement auprès du Trésor Public.

Une fois la situation comptable validée en collaboration avec le Trésor Public, la secrétaire clôt l'échéance sur le logiciel, ce qui à pour effet d'interdire tout nouveau

paiement ou modification des paiements sur cette échéance, et ouvre les paiements pour la suivante.

États comptables

À tout moment, la secrétaire peut générer différents états comptables, qui répertorient les paiements et les impayés de chaque échéance de l'année en cours. Ceux de l'échéance en cours sont donc sujets à évolution, tandis que ceux des échéances closes sont fixés.

Classes tarifaires et modes de paiements

La secrétaire définit dès le début de l'utilisation du logiciel l'ensemble des classes tarifaires qu'elle souhaite utiliser. Ces classes tarifaires correspondent à un critère déterminant un sous ensemble des élèves (étudiant, enfant ou adulte par exemple).

Des modes de paiements peuvent également être ajouté (ex : Chèques, carte bleue ...), ils sont qualifiés avec un montant maximum et un montant minimum autorisé.

Années

Au début de l'année, la secrétaire indique au logiciel les dates de début et de fin de l'année scolaire, ainsi que les dates des différentes échéances trimestrielles. Elle indique également une grille des tarifs pour l'année, avec pour chaque couple (prestation,classe tarifaire) des tranches qui correspondent à un quotient familial minimum, le montant maximum étant déterminé par le quotient minimum de la tranche strictement supérieure. La grille tarifaire pour l'année peut être modifiée tant qu'aucune inscription n'a été faite. Le passage d'une année à l'autre ne peut se faire que si toutes les échéances de l'année en cours sont closes. Ce passage clôt l'année et ouvre la première échéance de l'année suivante.

2.4.2 Formalisation avec MERISE

La description des processus métiers n'a pas été le cœur de l'analyse, nous avons souhaité privilégier une approche centrée sur les données et les fonctionnalités. C'est pourquoi mon travail s'est concentré sur le modèle conceptuel des données[1], toutefois, une brève analyse du contexte et des flux avec le modèle de contexte et le modèle de flux conceptuel permettront au lecteur d'appréhender les interactions entre le domaine étudié (la gestion de la facturation) et les deux acteurs externes qui sont les élèves et la gestion comptable.

Modèle de contexte (cf FIG-1, page 8)

Modèle de flux conceptuels (cf FIG-2, page 8)

FIG. 1 – Modèle de contexte

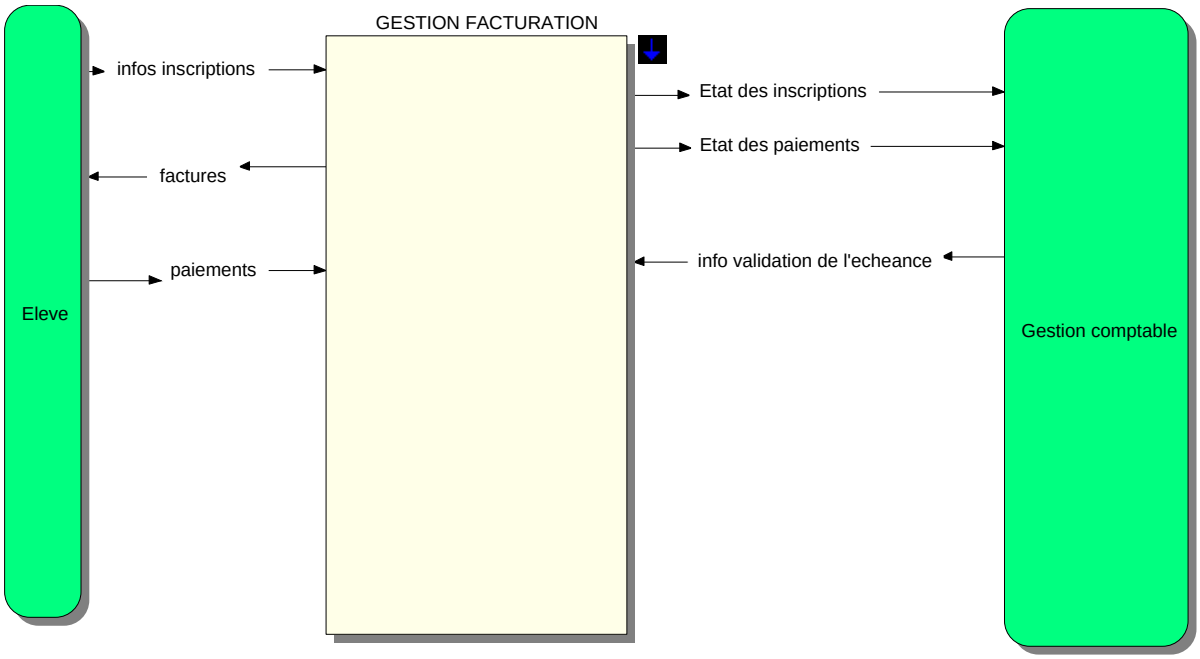
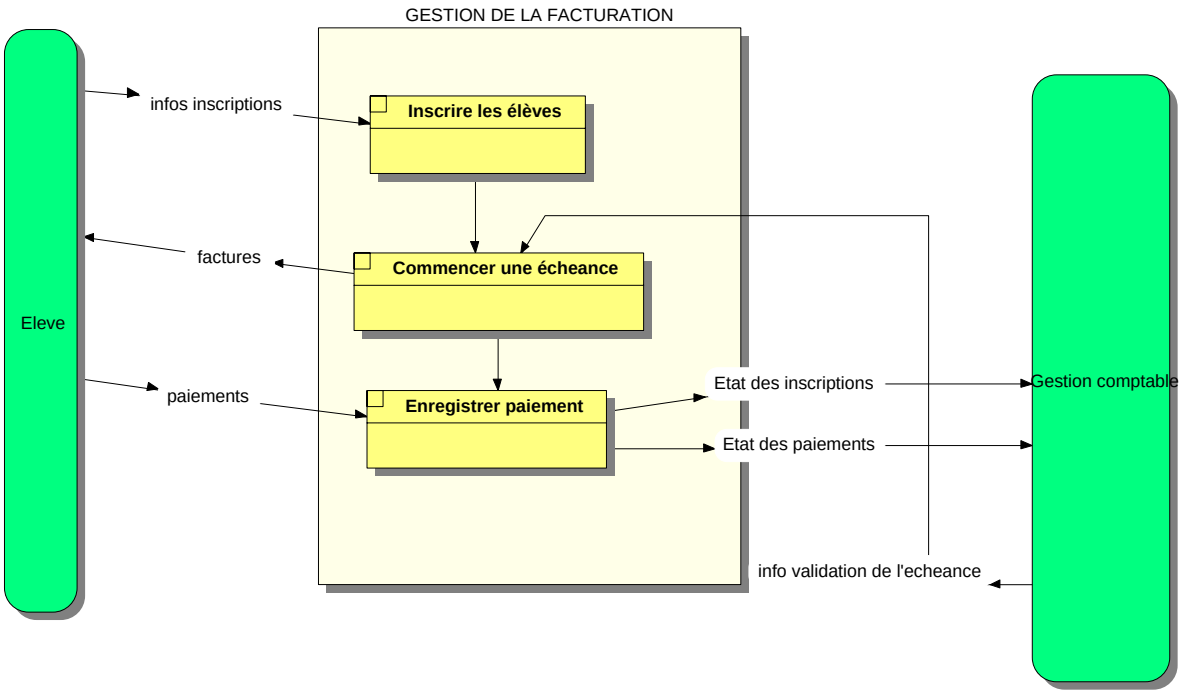


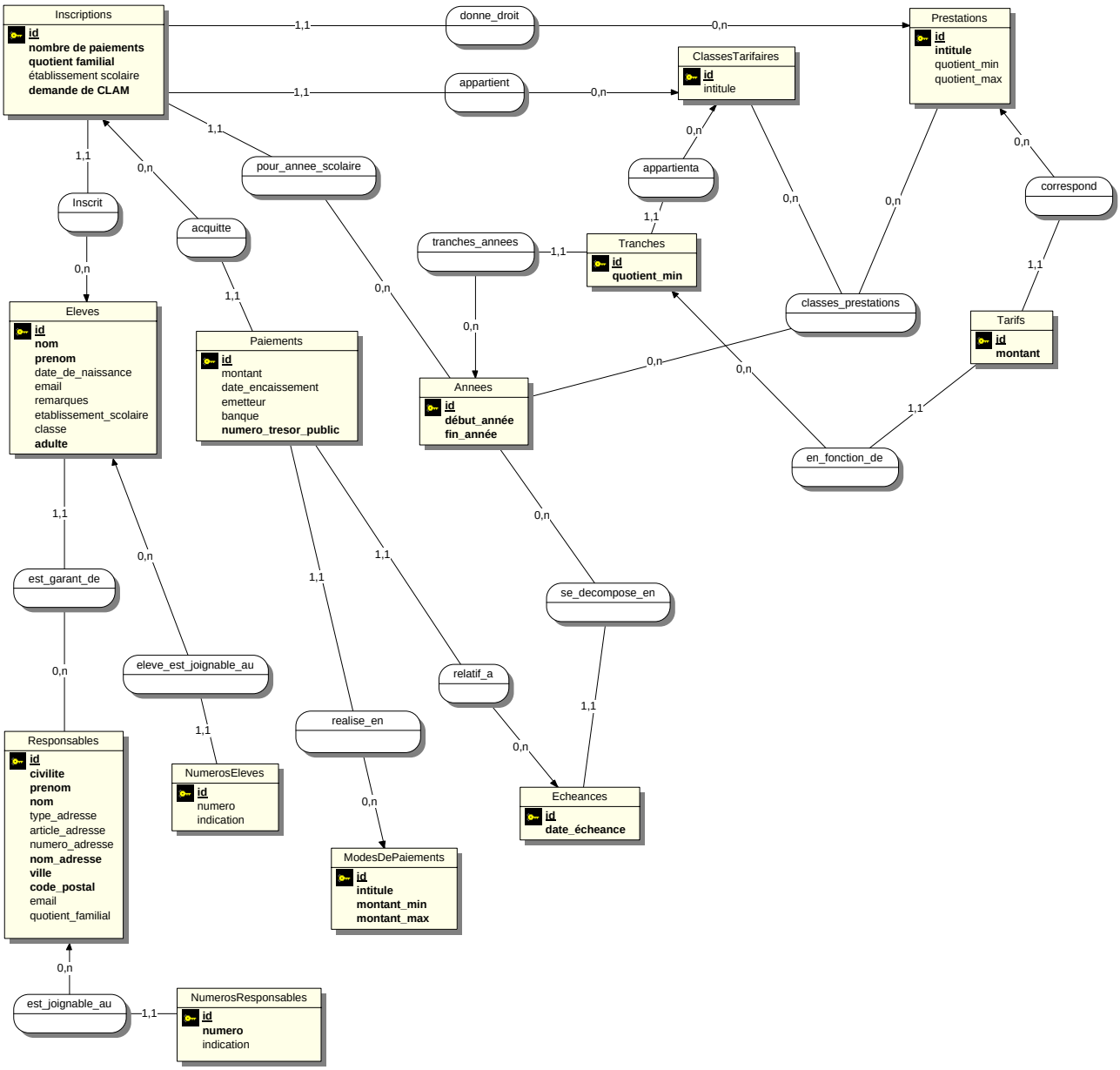
FIG. 2 – Modèle des flux conceptuels



Modèle conceptuel des données

L'étape majeure de l'analyse a été de créer un modèle conceptuel des données, celui-ci allant nous permettre de définir la base de données et les différents objets de notre application. Le schéma est donné en FIG-3, page 10.

FIG. 3 – Modèle conceptuel des données



3 Solution envisagée

3.1 Une approche projet centrée utilisateur

Afin de ne pas reproduire les défauts apparus dans la première version du logiciel, il a été décidé d'utiliser une méthode de développement un peu différente. Naturellement, les premiers jours du stage, j'ai utilisé la démarche de conception orienté activité enseignée à l'IUT2 (Définition des cas d'utilisation, diagramme de séquence de haut niveau puis détaillé pour chacun des cas...). Or je me suis vite aperçu que cette méthode ne serait pas adaptée au projet. Les besoins des utilisateurs et la formalisation de leurs processus métiers allant se faire au fur et à mesure du stage, je n'avais pas la matière nécessaire *à priori* pour mener une telle démarche.

De plus l'aspect ergonomique étant extrêmement important, je devais commencer d'abord par créer des interfaces qui allaient être discutées et validées avec les utilisateurs.

Plutôt que de partir d'une analyse des processus pour ensuite remonter aux interfaces, j'ai choisi de partir des interfaces pour en déduire les processus mis en œuvre. Cette démarche que l'on pourrait qualifier de *top-down*, permet d'éviter que les interfaces soient conçues avec un point de vue de technicien et d'informaticien, alors qu'elles doivent l'être du point de vue utilisateur.

Pour satisfaire à cette contrainte, une démarche un peu plus itérative a été menée. Des rendez-vous hebdomadaires avec l'école de musique ont été fixés afin de pourvoir faire évoluer le projet tout au long du stage et faire intervenir au maximum les utilisateurs, sans pour autant dépasser les limites fixés.

Le planning effectif du projet, en annexe A.5, page 47 illustre en partie cette démarche.

3.1.1 Approche fonctionnelle

La précédente version du logiciel était tournée essentiellement sur les processus, ce qui lui donnait une certaine rigidité. Afin de pallier à cet effet secondaire, l'architecture du projet à été envisagée sous un angle fonctionnel. Au lieu de définir des processus dans lesquels des fonctionnalités allaient êtres présentes, cette fois-ci ce sont les fonctionnalités qui ont été définies en premier, puis intégrées ensuite à une logique de processus.

En effet, les utilisateurs n'ont pas toujours formalisé leurs processus métiers, c'est le projet lui même qui lors de son déroulement vient les formaliser. En revanche, les utilisateurs sont très conscients dès le début du projet des fonctionnalités dont ils ont besoins. Cette approche a un principal avantage : les fonctionnalités peuvent être réorganisées facilement si les utilisateurs décident de modifier ou bien d'ajouter au cours du projet des processus métiers (meilleure réutilisabilité).

3.1.2 Ergonomie et utilisabilité

Critères à respecter

Afin de satisfaire aux exigences ergonomiques, il a fallu remplir trois critères : d'une part la satisfaction des utilisateurs à l'égard du logiciel (Esthétique, confort d'utilisation, appréciation subjective de l'utilisateur sur le logiciel), son efficacité (le logiciel me permet t'il de faire correctement ce que je souhaite ?) et l'efficience (le logiciel me permet t'il de faire ces actions rapidement et avec un moindre effort ?).[7]

Tout au long du projet, il a donc fallu garder à l'esprit ces trois critères.

Pour le critère d'efficacité, c'est l'approche fonctionnelle du projet qui a garanti son respect. En effet, la conception étant dictée par les besoins fonctionnels des utilisateurs, le risque de proposer des fonctionnalités inutiles ou bien qu'il manque des fonctionnalités indispensable est faible.

La satisfaction, critère plus subjectif, est prise en compte avec la volonté de proposer une interface graphiquement conviviale à l'utilisateur et graphiquement cohérente.

Enfin l'efficience est atteinte en partie avec une réflexion centrée sur les tâches que l'utilisateur a à accomplir et sur du simple bon sens. À noter aussi l'utilisation de modes d'interactions utilisateur un peu plus perfectionnés que ceux utilisés habituellement en environnement WEB (grâce notamment à l'utilisation de la technologie AJAX[2]pour certains éléments d'interface).

Organisation des fonctionnalités

Afin d'organiser au mieux les fonctionnalités, elles ont été regroupées par tâches. J'ai donc identifié quatre tâches principales qui sont l'inscription des élèves, la facturation, l'édition de documents, et la configuration du logiciel. L'arbre qui représente la carte du site découle directement de ce regroupement par tâches (cf FIG-4, page 13).

Participation des utilisateurs

Une des premières choses a été de proposer aux utilisateurs des maquettes compréhensibles pour des non informaticiens, commentées et réalistes des vues du logiciel, ceci afin de leur permettre d'apporter leur contribution à la définition des vues. Outre les considérations ergonomique, c'est à partir des vues définies en collaboration avec les utilisateurs que le besoin est définit en grande partie.

J'ai donc réalisé plusieurs maquettes avec le logiciel GIMP, qui ont ensuite évoluées au fur et à mesure des demandes de l'école de musique. En voici un exemple FIG-5, page 14.

[2]Voir Glossaire

FIG. 4 – Arborescence simplifiée de l'application

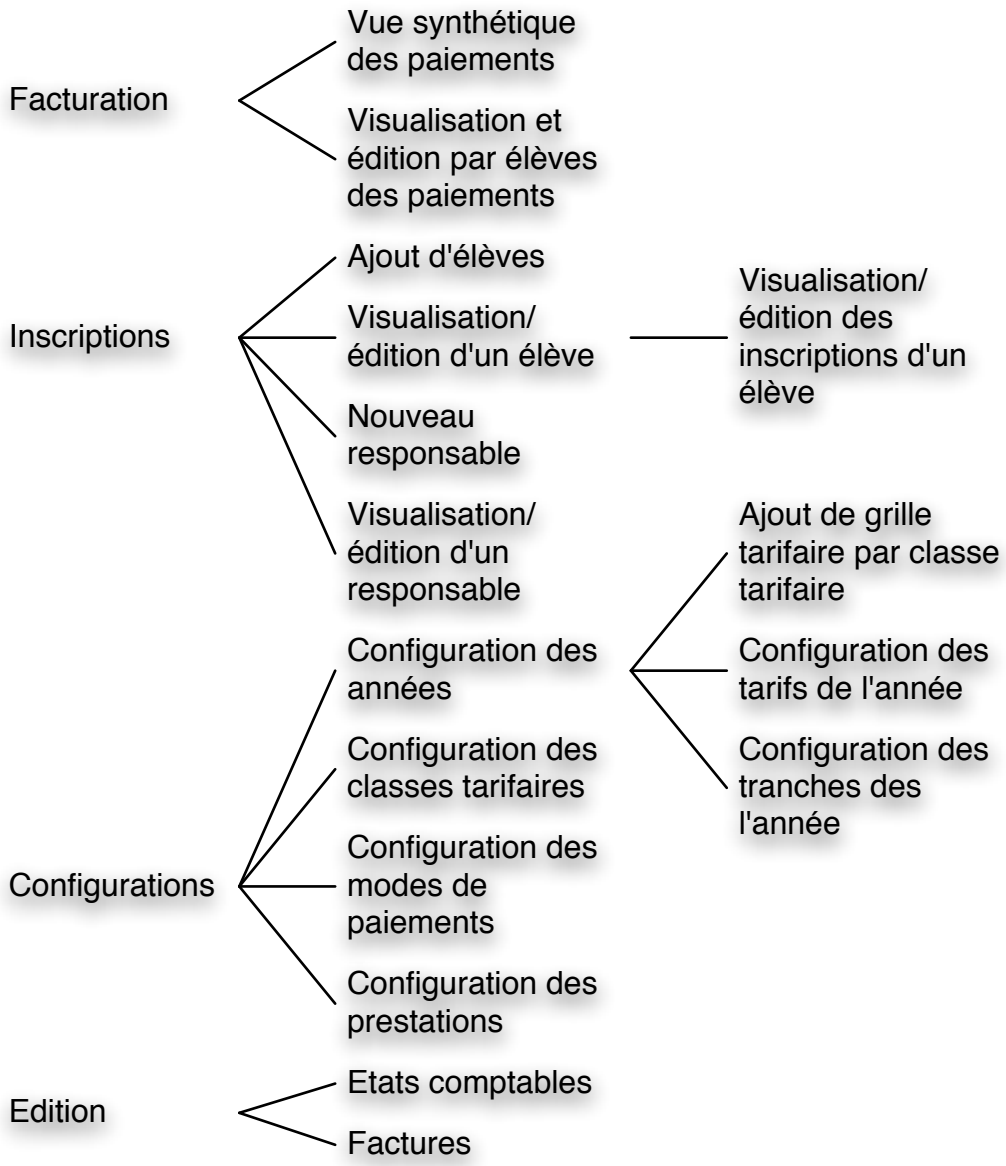
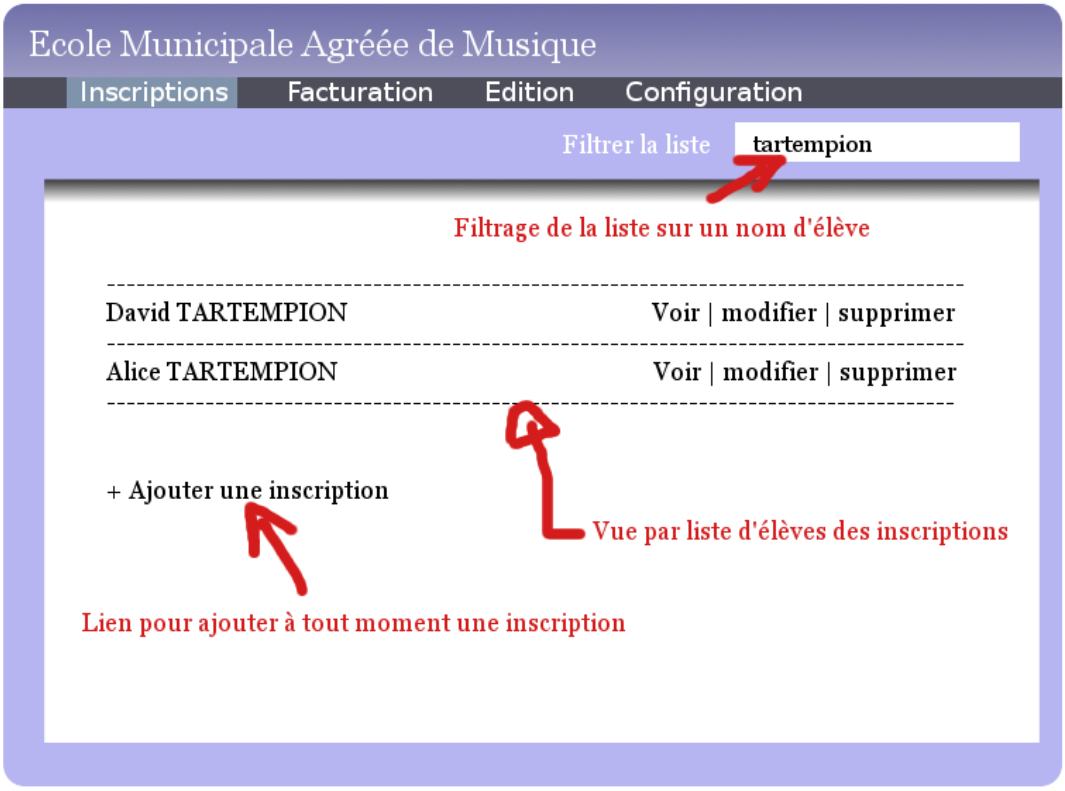


FIG. 5 – Maquette d'une vue



3.2 Choix techniques et technologiques principaux

3.2.1 Utilisation d'un framework orienté MVC : CakePHP

Une formation d'une journée nous à été proposée par l'ADDULACT afin de nous familiariser avec le Framework[3]CakePHP[6], que la DSI aimerait adopter pour une partie de ses futurs développements.

Ce framework, inspiré directement de Ruby On Rails[5], favorise les bonnes pratiques de programmation et les démarches de développement de type AGILE[4], dont la gestion de ce projet est inspirée.[3]

L'architecture MVC

Une des raisons du choix de ce framework est qu'il est très fortement orienté sur une architecture MVC, c'est-à-dire Modèle-Vue-Contrôleur. Ce patron de conception, mis au point en 1979 par Trygve REENSKAUG lors de la conception de SmallTalk[5]dans les laboratoires de XEROX PARC, propose une solution avec trois couches pour les applications avec interface utilisateur[8] :

- Les vues : L'utilisateur interagit directement avec cette couche, qui n'effectue aucuns traitements si ce n'est ceux dédiés à l'affichage. En général elles sont réalisées avec des langages de présentations (HTML par exemple). Elles ont pour seul rôle de présenter les données du modèle, en affichant ou masquant certaines informations.
- Les modèles : Ce sont les objets métiers de l'application. Ils se chargent de fournir les données avec leurs traitements associées et assurent leur accès et leur persistance.
- Les contrôleurs : ils se chargent de synchroniser les vues avec les modèles, et appellent les méthodes des modèles en fonction de la logique métier, pour renvoyer ensuite à l'utilisateur les vues correspondantes.

L'avantage de ce patron de conception est qu'il rend le code produit d'une très grande clarté, et favorise la réutilisabilité ainsi que la maintenabilité. On peut par exemple passer très facilement d'une vue sous forme de page WEB à une vue sous forme de fichier PDF.

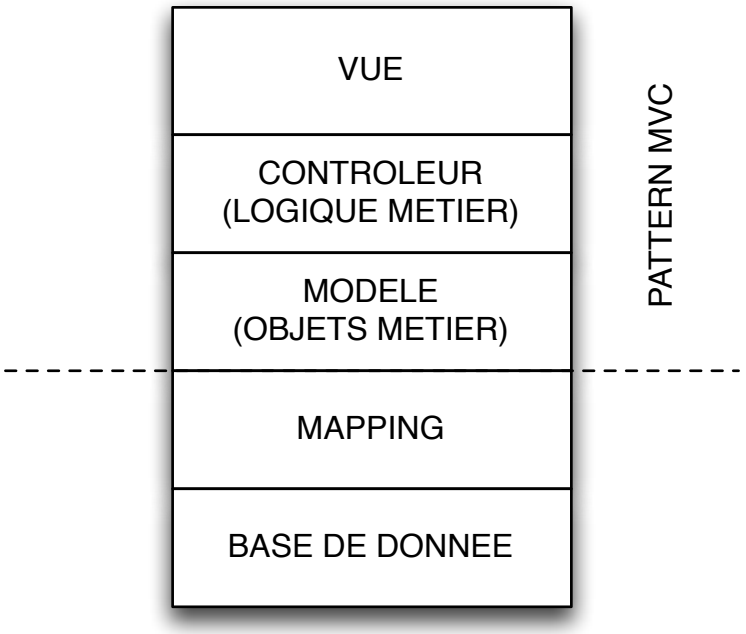
Le MVC appliqué au WEB

L'application du patron MVC aux technologies WEB a souvent été très problématique jusqu'à très récemment, en grande partie à cause de la primauté des architectures trois-tiers de types bases de données + procédure + interface, dans laquelle la logique objet ne trouvait pas vraiment sa place. Ce problème est désormais résolu grâce à l'ajout de

[3]Voir Glossaire
[4]Voir Glossaire
[5]Voir Glossaire

deux couches supplémentaires au patron MVC. Une base de données classique est toujours utilisée pour assurer la persistance des données, mais une couche *mapping* vient transformer les appels aux modèles en appels compréhensibles par la base de données, ce qui nous donne désormais une architecture cinq-tiers (cf FIG-6, page 16).

FIG. 6 – Architecture 5-tiers de CakePHP



Le MVC avec cakePHP

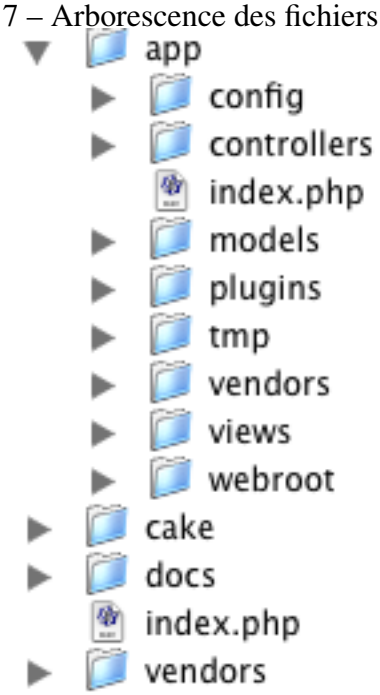
La clarté du patron MVC apparaît tout de suite au seul regard de l'arborescence proposée par CakePHP (cf FIG-7, page 17). Les dossiers models, views et controllers permettent d'accueillir respectivement les fichiers concernant les objets métiers, les vues et les contrôleurs. Lors de la programmation cette organisation est indéniablement efficace, et permet de s'y retrouver très facilement.

Mise en œuvre du patron MVC

Afin de pouvoir échanger des données entre les vues et les contrôleurs, le framework CakePHP propose la solution suivante : lors de l'appel par l'utilisateur d'une action du contrôleur, celui-ci récupère les données via les modèles, les empaquette, puis appelle la vue avec ces données. À charge ensuite à la vue de dépaqueter les données, puis de les afficher à l'utilisateur (FIG-8, page 18)

Lors d'une saisie d'un utilisateur, le mécanisme contraire est utilisé, c'est la vue qui empaquette les données, puis les envoie au contrôleur via une requête HTTP de type POST au contrôleur.

FIG. 7 – Arborescence des fichiers de CakePHP



Cette manière d'échanger les données est une utilisation du patron MVC assez spécifique au WEB, en effet, on préfère généralement en génie logiciel donner un accès en lecture des vues sur les modèles, et utiliser un mécanisme d'observation du modèle par les vues afin de les rafraîchir. Ceci est relativement difficile à réaliser en environnement WEB ou les échanges de données sont essentiellement synchrones.

Conventions de nommage de CakePHP

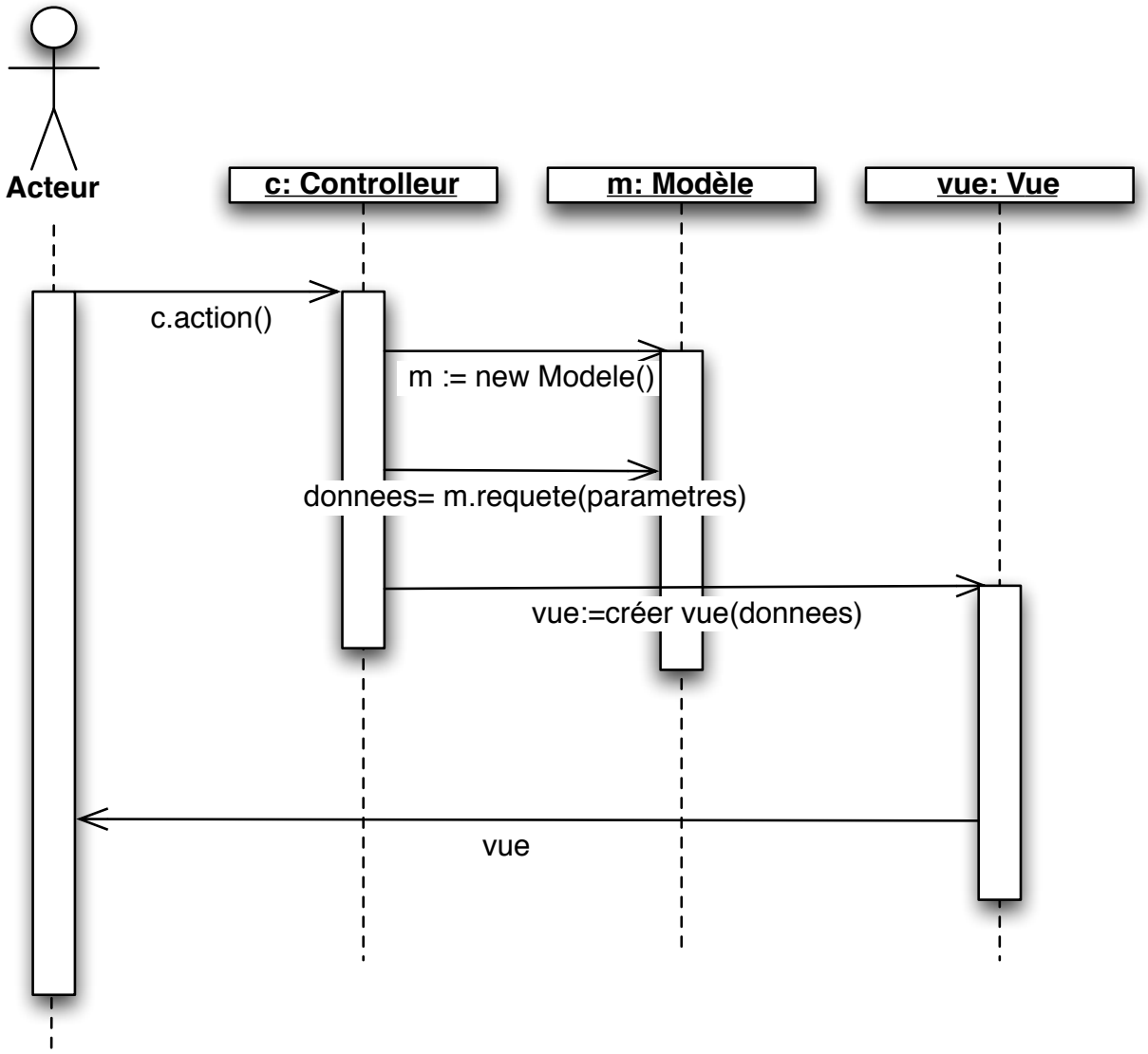
Un des aspects de cakePHP est qu'il propose un système bien défini de convention de nommage pour les différentes classes de l'application et la base de données.

Exemple : si l'on crée une classe modèle nommée "Inscription", alors la table dans la base de données devra s'appeler "inscriptions", la classe contrôleur associée à ce modèle sera nommée "inscriptions_controller", et les vues associées seront placées dans un sous dossier "inscriptions" du dossier views.

Ces conventions favorisent encore la lisibilité du code et sa réutilisation, le nom des fichiers uniquement permet de comprendre instantanément sa fonction au sein de l'application. Mais surtout, et c'est le plus intéressant, le framework utilise ces conventions pour établir une partie de la logique applicative.

Pour illustrer cet aspect, voici un exemple de problème que l'on rencontre avec du code rédigé en langue française : si l'on a une table Eleve dans la base de données, alors le nom du modèle attendu sera "Elefe" et non "Eleves", puisque que CakePHP utilise le système de déclinaisons singulier/pluriel de la langue anglaise. Ceci peut bien

FIG. 8 – Utilisation du paradigme MVC avec CakepHP



sûr être corrigé grâce à une configuration un peu plus fine du système des déclinaisons singulier/pluriel, mais montre bien à quel point CakePHP tire parti jusqu'aux subtilités du langage naturel pour rendre le développement plus rapide.

CakePHP propose également des conventions au niveau des attributs de la base de données : un entier auto-incrémenté nommé id est préconisé comme clef primaire pour toutes les tables, et les clefs étrangères se composent de la manière suivante : nomdelatable_id. Ceci implique donc souvent des modifications au niveau de la base de données si l'on veut profiter des avantages apportés par ces conventions.

Composition des modèles

Afin d'établir les relations de compositions entre les objets et assurer le "Mapping" des objets métiers vers la base de données, CakePHP offre quatre types de relations entres les classes modèles :

- belongsTo : Relation de cardinalité 1,1
- hasOne : Relation de cardinalité 0,1
- hasMany : Relation de cardinalité 0,n
- hasAndBelongsToMany : Relation de cardinalité 0,n - O,n

Bien que limité, ce système permet néanmoins d'établir très rapidement un modèle de données qui va bénéficier des possibilités de la super-classe Modèle de CakePHP. Celle-ci offre en effet de nombreuses méthodes d'accès aux données qui vont pouvoir être utilisées dans les contrôleurs.

3.3 Mise en œuvre du framework CakePHP pour l'école de musique

3.3.1 Prototypage

CakePHP est parfaitement adapté à l'approche centrée utilisateur et itérative que nous avons décidé de privilégier pour le projet. En effet ce framework dispose de fonctionnalités de prototypage rapide qui ont permis, dès la construction de la base de données, de proposer des maquettes fonctionnelles à l'école de musique. Une de ces fonctionnalités de type RAD[6]est le *scaffolding*, que l'on pourrait traduire par échafaudage. Après la construction du modèle de donnée, il suffit en effet d'indiquer la variable *scaffolding* dans le contrôleur souhaité, et l'application est tout de suite capable d'effectuer les opérations de bases CRUD (Create, Read, Update, Delete) qui permettent de créer, lire, modifier ou détruire un objet du modèle. À partir des champs de la base de données et du modèle relationnel défini dans les classes Modèle, le *scaffolding* génère dynamiquement à l'exécution les actions du contrôleur et les vues qui vont permettre de mettre en œuvre ces fonctionnalités de base.

Bien sûr la plupart du temps, ces fonctionnalités ne suffisent pas et n'intègrent pas la logique métier, mais elles permettent de créer de manière immédiate des prototypes

[6]Voir Glossaire

de l'application destiné à l'utilisateur, qui peut ainsi suivre de manière concrète l'avancement du projet et faire part de ses besoins au fur et à mesure.

3.3.2 Génération de code

Nous avons vu les avantages du prototypage rapide, mais il a aussi ses limites. En effet, les fonctionnalités de bases proposées par le *scaffolding* sont nécessaires mais doivent être complétées pour notre projet.

Aussi, pour continuer à utiliser les avantages du *scaffolding* tout en permettant une personnalisation totale des différentes actions, un outil de génération de code en ligne de commande nommé "Bake" est intégré à CakePHP. Cet utilitaire permet notamment de créer de façon interactive le modèle de données à partir de la base SQL, puis les contrôleurs avec le code de leurs actions "CRUD", ainsi que pour chaque contrôleur les vues sur ces actions. Le modèle de données généré par Bake après l'analyse de la base de données est présenté FIG-9, page 22.

Sont aussi présentés ici des extraits du code des modèles, vues, et contrôleurs générés par Bake pour la classe ModeDePaiement, qui définit les différents modes de paiements que peuvent utiliser les élèves.

Modèle d'un mode de paiement

```
class Modedepaiement extends AppModel {

  //definition d'une relation 0,n vers la classe
     paiement
  var $hasMany = array(
      'Paiement' => array('className' => 'Paiement')
  );

}
```

Action "add()" du contrôleur mode_de_paiements

```
  function add() {
    if(!empty($this->data)) { //si le controlleur a
       recu des donnees
      $this->cleanUpFields(); //reformatage des champs
          pour les rendre conforme a la BDD
      $this->Modedepaiement->create(); //instanciation
          de l'objet
      if($this->Modedepaiement->save($this->data)) {
         //sauvegarde des donnees
        //message de confirmation et redirection sur
           une autre vue
```

```
            $this->flash('Mode de paiement enregistre',
               array('action'=>'index'));
            exit(); //sortie de la procedure
          } else {
          }
        }
        //implicitement la vue add de mode de paiement est
            affichee.
    }
```

Vue associée à l'action "add()" du contrôleur mode_de_paiements

```
<div class="modedepaiement">
<h2>Nouveau Mode de paiement</h2>
  <?php echo $formFrench->create('Modedepaiement');?>
    <?php echo $formFrench->input('intitule');?>
    <?php echo $formFrench->input('montant_min');?>
    <?php echo $formFrench->input('montant_max');?>
    <?php echo $formFrench->submit('Ajouter');?>
  </form>
</div>
```

3.3.3 Les interfaces utilisateurs

Un des avantages du MVC est de pouvoir utiliser différents types d'interfaces pour un même modèle. C'est exactement ce qui à été mis en œuvre dans ce projet. Par exemple, la secrétaire de l'école de musique souhaitait visualiser l'état des paiements de ses élèves avec une interface WEB conviviale et intuitive (cf FIG-10, page 23), mais souhaitait également disposer de factures au format PDF qu'elle pourrait imprimer elle même, ainsi que de différents états comptables au format Excel à destination du Trésor Public.

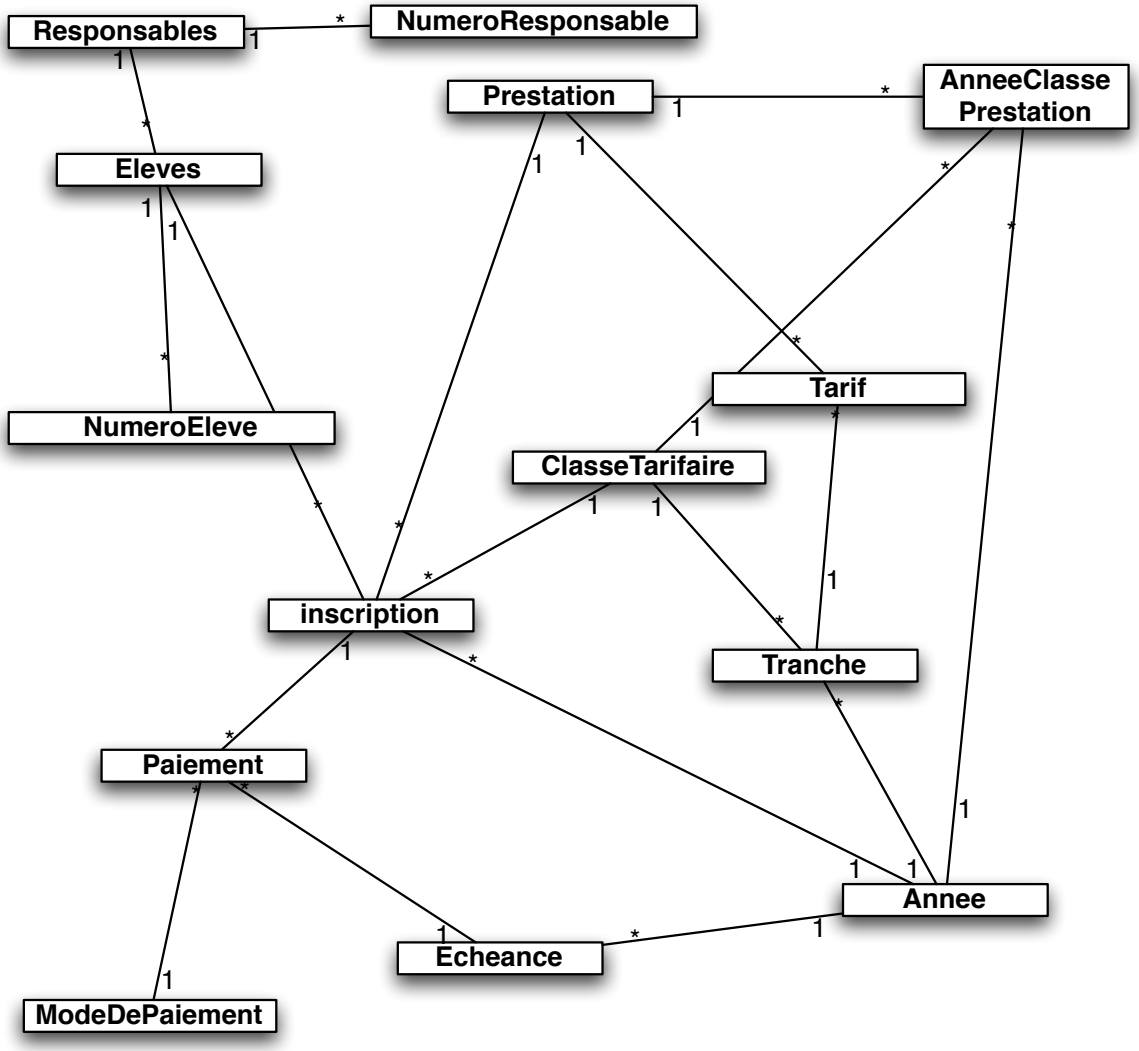
Cohérence graphique

Afin d'avoir une charte graphique cohérente, l'utilisation de feuilles de styles CSS[7]à été privilégiée[2]. Cela permet de garantir un formattage unique de tous les éléments du même type sur toutes les pages HTML.

Des copies d'écrans du logiciel sont disponibles en annexe 10 page 23 et illustrent cet aspect.

[7]Voir Glossaire

FIG. 9 – Diagramme de classe simplifié du modèle

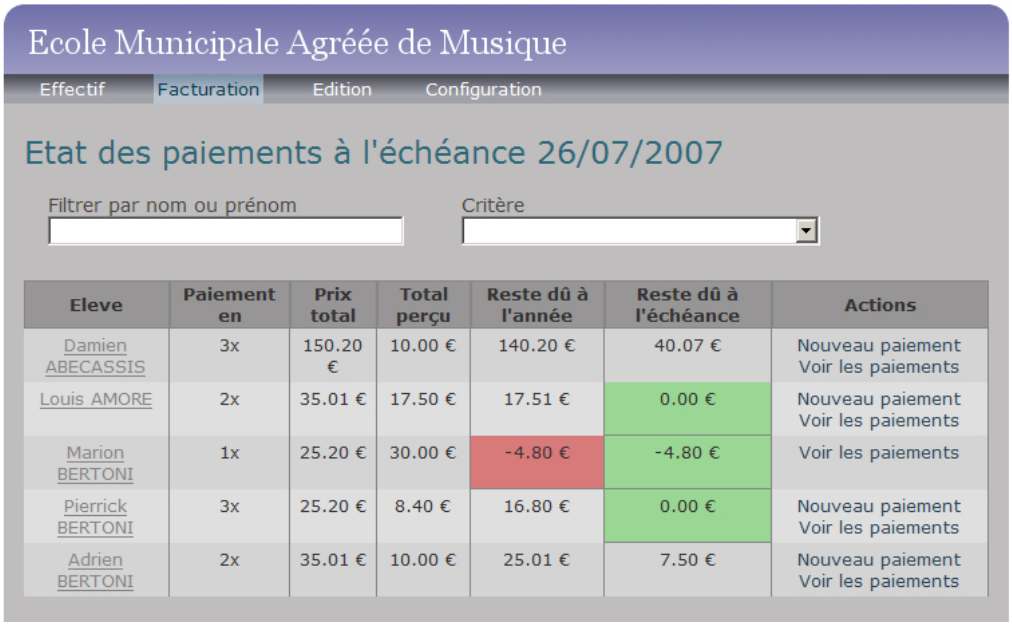


Afin de faciliter l'utilisation d'interfaces multiples, CakePHP dispose d'un système de *helpers*. Ces *helpers* sont en fait des classes qui fournissent un ensemble de fonctionnalités permettant de présenter les données reçues par les vues selon un format désiré. Il existe par exemple un helper HTML, un helper Javascript, un helper XML[8], et bien d'autres encore. Une des possibilités offerte au développeur est de créer ses propres *helpers*, ce qui à été fait pour ce projet pour la génération de fichiers Excels et PDF, à partir d'exemples fournis par le site Bakery[4].

Lorsqu'un contrôleur appelle une vue, il peut lui associer un *layout*, c'est à dire un cadre fixe dans lequel le contenu va venir s'inscrire. Dans le cas d'une page HTML il s'agira de la structure HTML de la page ainsi que de la feuille de style CSS, dans le cas

[8]Voir Glossaire

FIG. 10 – Vue des paiements



d'une page PDF ou Excel il s'agira de la redéfinition des entêtes HTTP nécessaires pour le téléchargement.

3.3.4 Base de données MySQL

La base de données disponible au niveau de la DSI est MySQL 5.

Afin de construire la base de données et y accéder, j'ai utilisé essentiellement l'utilitaire phpMyAdmin[9] que la DSI utilise régulièrement. Cette version de MySQL présente l'avantage de gérer de façon basique les vues, les procédures stockées, les *triggers*[9]et les clés étrangères. Toutes ces possibilités ont été utilisées pour la réalisation du logiciel.

Optimisations

Une des principales raisons d'utiliser des vues et des procédures stockées pour certaines opérations est liée au déficit de performance engendré par le système de *mapping* du modèle objet vers la base de données. Lorsque l'on souhaite par exemple donner une vue synthétique des paiements à l'utilisateur, il faut récupérer les informations relatives à chaque objet dont les données entrent dans la composition de cette vue. Or, le *mapping* récursif, qui consiste à, pour un objet, récupérer les données de ses objets liés, devient vite très coûteux en mémoire et en temps d'exécution. En mémoire, car certains modèles liés sont inutiles dans le cas d'utilisation, et en temps, car CakePHP va multiplier de manière impressionnante le nombre de requêtes faites à la base de données.

[9]Voir Glossaire

En effet le nombre de requêtes effectuées est exponentiel avec la profondeur de récursivité. Ainsi, pour joindre trois tables contenant 100 éléments chacune, cakePHP peut être amené suivant les cas à faire 1x100x100 soit 10000 requêtes SQL, ce qui au mieux prend un temps fou, au pire met le serveur MySQL à genoux.

C'est là que l'utilisation de vues et de procédures stockées intervient. Grâce à une vue MySQL, il est possible de créer une table dynamiquement mise à jour à partir d'une requête complexe portant sur plusieurs tables, et qui de plus est mise en cache dans la base de données, ce qui assure un temps d'accès moyen à la vue extrêmement favorable. Au niveau du modèle de données objet CakePHP, il suffit alors de définir une méthode d'accès sur cette vue MySQL dans le modèle concerné. Le code d'une vue sur la base est présenté en annexe A.1 page 31.

Les procédures stockées, quant à elles, permettent de répondre à des demandes spécifiques et ponctuelles de valeurs calculées à partir de plusieurs requêtes complexes (cf A.3 page 35).

Gestion de la cohérence des données

CakePHP propose bien sûr de gérer la cohérence des données via les relations définies dans le modèle objet. Toutefois, il est apparu indispensable de garantir cette cohérence au niveau même de la base de données, c'est en effet le seul moyen de garantir réellement cette cohérence, même en cas de bogue de l'application ou de défaillance du serveur d'application.

Pour cela les contraintes de clefs étrangères apportées par le moteur de stockage InnoDB dans MySQL ont été d'un grand secours, ainsi que l'utilisation de *triggers*. Ceux-ci permettent en effet de gérer des contraintes complexes. L'application de gestion de l'école de musique permet notamment de gérer des grilles de tarifs (A.4.2 page 45). Ce sont des *triggers* qui ont permis de faire en sorte que lorsqu'une colonne ou une ligne est ajoutée à la grille, les cellules correspondantes (tarifs) soient automatiquement ajoutées à la base de données, et supprimées lors de la suppression de colonnes ou de lignes. Le code de ces *triggers* est fourni en annexe A.2 page 33.

Bien sûr il faut prendre garde, dans la mesure du possible, à ne pas transférer certains aspects spécifiquement métiers sur la base de données. Certaines contraintes et procédures sont spécifiquement liées aux données, et d'autre sont dépendantes de la logique métier.

Ce sont ces contraintes et procédures liées aux données qui ont été intégrées à la base SQL, les autres doivent rester implémentées au niveau de l'application, car c'est le seul moyen de bien séparer ces deux aspects.

4 Intégration et mise en œuvre de la solution

4.1 Intégration au contexte technique

4.1.1 Forge de l'addulact

L'ADDULACT met à la disposition des développeurs des collectivités une plateforme de développement collaboratif. Cette *forge* propose entre autre la possibilité d'utiliser le système de gestion des sources CVS[10]. Ce système permet de conserver sur un serveur l'historique de toutes les modifications faites sur le code source, et éventuellement de gérer les ajouts de différents développeurs sur un même fichier source.

Bien qu'étant seul sur le projet, et donc l'aspect collaboratif de CVS peut intéressant à cet égard, j'ai bien compris l'intérêt de ce système notamment lors de l'utilisation de tests de non-régression. À chaque modification sur le code, si le test de non-régression échoue alors l'historisation effectuée par CVS permet de visualiser très facilement les changements qui sont à l'origine de l'erreur, et éventuellement de revenir à une version précédente.

Afin de gérer correctement le CVS, nous avons utilisé l'environnement de développement intégré Eclipse, qui offre un module d'accès convivial au serveur CVS.

4.1.2 Sécurité

La DSI dispose d'un système de *login* de type SSO[11]pour toutes ses applications, et d'un annuaire LDAP[12]qui recense les différents comptes utilisateurs. Afin d'intégrer ce SSO au logiciel de l'école de musique, Philippe GODOT de Probesys, prestataire technique auprès de la DSI, est venu nous faire une courte formation à ce sujet.

Le principe du SSO est de permettre à une personne de s'authentifier une seule fois sur l'intranet pour avoir accès à toutes les applications qui s'y trouvent et sur lesquelles elle est autorisée à naviguer.

Afin de réaliser ceci, lorsque l'utilisateur appelle une des pages du logiciel, ce dernier vérifie automatiquement si une session est ouverte avec l'utilisateur, cette session étant partagée par toutes les applications de la DSI. Si ce n'est pas le cas, l'utilisateur est redirigé vers une page de *login*, qui se chargera de la vérification des identifiants de connexion via LDAP, puis créera la session si l'authentification est réussie.

Ce système, mis en place à l'essai avec l'aide de Probesys, sera implanté dans la version finale à la fin du stage.

4.1.3 Mise en place sur le serveur

Développé sur un poste informatique en local, la version de développement était toutefois régulièrement mise à disposition sur le serveur WEB de la DSI via FTP, afin de

[10]Voir Glossaire
[11]Voir Glossaire
[12]Voir Glossaire

permettre aux utilisateurs de se rendre compte de l'avancement du projet et d'effectuer des tests.

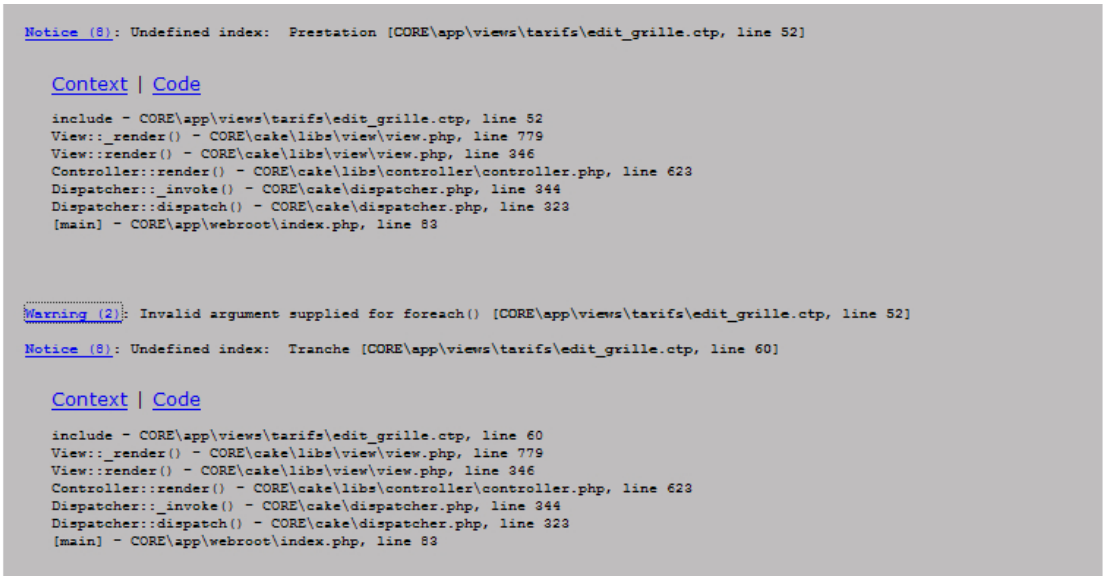
Pour la version finale, il est prévu d'effectuer une installation un peu plus complexe du framework CakePHP. Actuellement seule une application peut être utilisée avec le framework. Or d'autres stagiaires ayant également travaillé sur ce même framework, il peut être intéressant de disposer d'une seule version du *core* du framework pour toutes les applications, afin d'en faciliter la mise à jour.

4.2 Tests et corrections de bogues

4.2.1 Traces

CakePHP est équipé d'un débogueur intégré qui permet de détecter la majorité des bogues en affichant une trace très précise des actions effectuées, et indique les lignes de code incriminées. Plusieurs niveaux de débogage sont accessible, de 0 (erreurs masquées) à 3 (Traces complètes de tous les objets).

FIG. 11 – Débogueur intégré à CakePHP

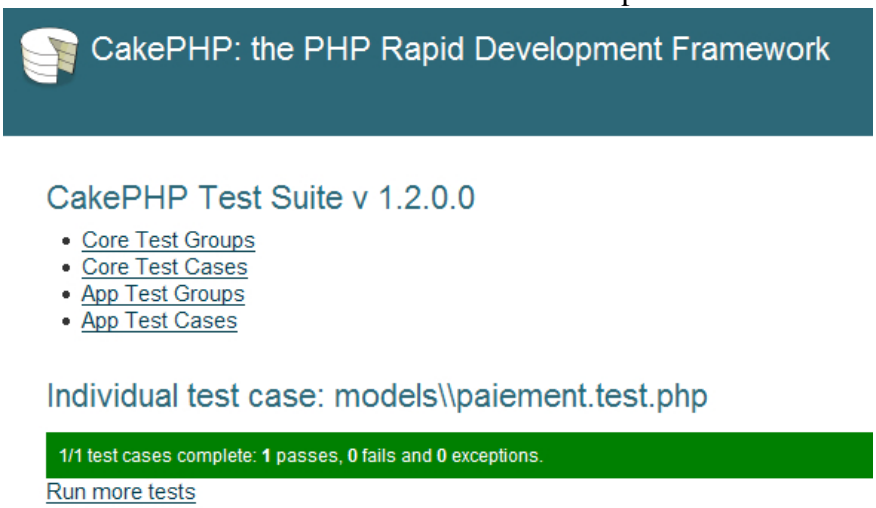


4.2.2 Tests de non-régression

Ce type de tests permet de vérifier que les modifications appliquées au logiciel en cours de développement n'ont pas induit un dysfonctionnement et que le logiciel fonctionne toujours de la même façon.

Pour effectuer ces tests, la librairie de tests unitaires SimpleTest[10] à été utilisée. Elle fournie une interface agréable pour lancer les tests. Chaque classe de test est composée d'une fonction *SetUp* (Création des objets nécessaires pour le test), *tearDown*

FIG. 12 – Interface de test avec SimpleTest



(suppression des objets en fin de test), et le programmeur peut y ajouter autant de fonction de test que nécessaire. De plus , le générateur de code "Bake" peut créer automatiquement la structure de ces fichiers de tests pour les modèles et les contrôleurs, que le développeur est libre de compléter.

4.2.3 Tests utilisateurs

Afin de garantir la fiabilité du logiciel, la secrétaire de l'école de musique s'est portée volontaire pour effectuer un ensemble de tests, notamment en simulant une année de saisie. Ces tests, en cours lors de la rédaction de ce rapport, sont au moins aussi importants que les tests de non-régression, car ils vont permettre de mettre en exergue non seulement d'éventuels bogues au niveau des interfaces, mais également des imperfections ergonomiques, particulièrement difficiles à détecter pour le concepteur du logiciel. En effet, il est surprenant de voir à quel point une phase de tests peut révéler que l'attitude de l'utilisateur face au logiciel n'est pas du tout celle que le concepteur aurait imaginée.

De plus ces tests pourront mettre en évidence les points sur lesquels le logiciel ne serait pas suffisamment intuitif et qui pourront donner l'occasion à un révision d'un partie de la logique de l'interface, ou dans le pire des cas faire l'objet d'une explication dans le manuel utilisateur, si les modifications s'avèrent impossibles dans le temps imparti.

4.3 Intégration au contexte humain et juridique

4.3.1 Contexte juridique

Le framework CakePHP est distribué sous licence MIT[13], ce qui signifie que sa licence d'exploitation peut être modifiée sans restrictions pourvu que soit indiquées certaines informations concernant les auteurs du framework dans le logiciel final.

La DSI a choisi d'utiliser la licence CECILL[14] pour le logiciel de gestion de l'école de musique, ce qui permettra la distribution et la modification du logiciel pour d'autre collectivités, tout en étant compatible avec le droit français.

4.3.2 Documentation

Une documentation développeur rédigée à l'aide du générateur de documentation de type Doxygen et un manuel à l'attention des utilisateurs sont prévus. Lors de la rédaction de ce rapport, ces documentations n'ont pas encore été finalisées, le stage étant toujours en cours. Toutefois ces documents seront présentés au jury le jour de la soutenance.

4.4 Améliorations possibles

4.4.1 Extensions possibles

Une éventuelle extension du projet serait d'adjoindre au logiciel des possibilités de suivi pédagogique des élèves. Actuellement ce suivi est réalisé indépendamment du logiciel de facturation, ce qui pourrait poser des problèmes de cohérence des informations entre la base de suivi pédagogique et celle du logiciel de facturation, et implique une duplication des saisies de certaines informations, notamment celles concernant les élèves.

De plus la direction de l'école de musique m'a fait remarquer que l'édition des bulletins scolaires posait problèmes, en effet, ils sont distribués sur papier et beaucoup de professeurs s'y prennent au dernier moment pour les remplir. Cela est source de perte de temps pour l'administration. Un bon moyen serait de proposer un système de remplissage et de publication en ligne des bulletins, que les professeurs pourraient remplir facilement depuis leur ordinateur et leur connexion internet personnelle.

4.4.2 Réutilisations possibles

Pour d'autres structures

L'utilisation du Framework CakePHP ainsi que la méthode de gestion de projet mise en œuvre a permis d'aboutir à un logiciel qui serait très facilement réutilisable pour d'autres structures du même type que l'école de musique, qui ont besoin de gérer une facturation liée à des inscriptions et fournissant un ensemble de prestations. Par exemple toutes les structures ludo-éducatives (écoles de dessin, de sport ...) mais aussi pourquoi pas les cantines scolaires ou d'autres structures du même type.

[13] Massachusetts Institute of Technology

[14] (Ce(A)C(nrs)I(NRIA)L(ogiciel)L(ibre))

Il serait donc intéressant d'identifier les aspects communs de ces différentes structures, puis de dégager du logiciel de gestion de l'école de musique uniquement le code correspondant à ces aspects, de telle manière à en favoriser la spécialisation.

Pour d'autres DSI

Afin de faciliter l'installation du logiciel pour d'autres écoles de musique, il est prévu de réaliser si le temps le permet un petit installeur, qui viendra automatiser certaines tâches indispensables (configuration du framework, création de la base de données ...)

Pour la DSI

Pour la DSI, de nombreux composants créés pour ce logiciel sont directement réutilisables pour d'autres projets, notamment le *helper* de génération de PDF ainsi que celui de génération de fichiers Excels, créé spécialement pour l'occasion.

Réutilisations des savoirs

Avec une formation d'une journée sur le framework CakePHP et deux mois de stage qui ont permis d'en appréhender toutes les facettes, il paraît dommage que les savoirs et savoir-faire acquis disparaissent de la DSI lors du départ des stagiaires qui ont travaillé sur ce framework.

Une amélioration serait donc de constituer une base de connaissances sur CakePHP ainsi que des patrons de processus[11] décrivant précisément les étapes du développement avec ce framework, afin que ces acquis puissent rester au sein de la DSI de Fontaine et être réutilisés pour des développements futurs.

5 Conclusion

Pour finir, ce stage a représenté une excellente expérience à bien des égards. En combinant aussi bien analyse, développement, et gestion de projet, ce sujet m'a permis d'aborder les principales caractéristiques de toutes les étapes permettant de mener à bien ce type de mission.

Une phase d'analyse tout d'abord m'a donnée l'occasion de faire ressortir les connaissances acquises en DUT dans ce domaine et de les mettre en pratique. Cette mise en pratique sur le terrain de ces outils d'analyse m'a permis de prendre du recul par rapport à ceux-ci. J'ai compris notamment que la maîtrise de ces outils ne suffisait pas, les utiliser de façon mécanique et rigide ne menant à rien sur ce type de projet. Seule une approche pragmatique du problème et une utilisation pertinente et bien ciblée des outils d'analyse permet de réduire au maximum la charge du projet tout en assurant sa bonne conduite.

Puis une phase de codage assez longue m'à également donnée l'occasion de me servir de mes connaissances de DUT dans ce domaine, et de les perfectionner, notamment en matière de bases de données relationnelles, de programmation WEB, et d'utilisation de framework.

Dans la gestion du projet, j'ai tenté d'apporter non seulement les acquis de l'IUT, mais aussi, grâce à la relative autonomie dont je disposais, un style un peu plus personnel. J'ai opté pour une approche centrée utilisateur qui s'est révélée payante, puisque le projet a pu être mené à terme.

D'un point de vue professionnel, ce stage a rempli à mon sens ses objectifs, puisque j'ai pu mettre en application mes connaissances de DUT et les confronter à la réalité du terrain. J'ai pu avoir également un aperçu de la structure d'une direction des systèmes d'information et des réalités auxquelles un service de ce type doit faire face. Ce projet m'a exposé à des difficultés techniques auxquelles il a fallu répondre par une acquisition autodidacte de certaines compétences. Il a fallu être également capable de créer une relation avec les utilisateurs et comprendre leurs besoins en se mettant à leur place.

Sur le plan personnel, j'ai acquis grâce à ce stage une certaine confiance en mes connaissances, ce qui me permettra d'aborder par la suite des projets plus complexes avec plus de sérénité. De plus j'ai pu mesurer précisément l'étendue et les limites de mes capacités dans certains domaines.

Expérience personnelle que je pourrais valoriser, ce stage de deux mois fut une excellente opportunité riche en enseignements.

A Annexes

A.1 Exemple de vue sur la base de données : état des paiements

```
-- ----------------------------------------
-- Vue sur l'etat des paiements pour chaque annees et
   chaque echeance
-- ----------------------------------------
DROP VIEW IF EXISTS paiements_en_cours_par_echeance;
CREATE VIEW paiements_en_cours_par_echeance AS
(SELECT
    Annee.id as annee_id,
    Echeance.id as echeance_id,
    total_percu_avant_echeance(Inscription.id,Echeance
       .id) as total_percu,
    tarif_eleve_au(eleve_id,Annee.id)-
       total_percu_avant_echeance(Inscription.id,
       Echeance.id) as restant_du,
    du_encours_au(eleve_id,nombre_de_paiements,
       Echeance.id,Annee.id) as du_encours,
    du_encours_au(eleve_id,nombre_de_paiements,
       Echeance.id,Annee.id)-
       total_percu_avant_echeance(Inscription.id,
       Echeance.id) AS restant_du_encours,
    Eleve.id as eleve_id,
    Eleve.nom,
    Eleve.prenom,
    Responsable.quotient_familial AS qf,
    Responsable.civilite,
    CONCAT(Responsable.prenom,' ', Responsable.nom) as
        responsable,
    Responsable.complement_adresse as
       complement_adresse,
    CONCAT(Responsable.numero_adresse,' ',Responsable.
       type_adresse,' ',Responsable.article_adresse,'
       ',Responsable.nom_adresse) as adresse,
    CONCAT(Responsable.code_postal,' ',Responsable.
       ville) as ville,
    nombre_de_paiements,
    Inscription.id as inscription_id,
    Prestation.id as prestation_id,
    Prestation.intitule,
    tarif_eleve_au(eleve_id,Annee.id) as total_du
```

```
FROM
  responsables AS Responsable,
  inscriptions AS Inscription,
  eleves AS Eleve,
  prestations AS Prestation,
  echeances AS Echeance,
  annees AS Annee
WHERE
  Eleve.id=Inscription.eleve_id
  AND Responsable.id=Eleve.responsable_id
  AND Inscription.prestation_id=Prestation.id
  AND Inscription.annee_id=Annee.id
  AND Annee.id=Echeance.annee_id
  AND nombre_de_paiements>nb_echeances_finies_au(
     Echeance.id))
UNION
(SELECT
  Annee.id as annee_id,
  Echeance.id as echeance_id,
  0 as total_percu,
  tarif_eleve_au(eleve_id,Annee.id) as restant_du,
  du_encours_au(eleve_id,nombre_de_paiements,
     Echeance.id,Annee.id) as du_encours,
  du_encours_au(eleve_id,nombre_de_paiements,
     Echeance.id,Annee.id) as restant_du_encours,
  Eleve.id as eleve_id,
  Eleve.nom,
  Eleve.prenom,
  Responsable.quotient_familial AS qf,
  Responsable.civilite,
  CONCAT(Responsable.prenom,' ', Responsable.nom) as
      responsable,
  Responsable.complement_adresse as
     complement_adresse,
  CONCAT(Responsable.numero_adresse,' ',Responsable.
     type_adresse,' ',Responsable.article_adresse,'
     ',Responsable.nom_adresse) as adresse,
  CONCAT(Responsable.code_postal,' ',Responsable.
     ville) as ville,
  nombre_de_paiements,
  Inscription.id as inscription_id,
  Prestation.id as prestation_id,
  Prestation.intitule,
```

```
    tarif_eleve_au(eleve_id,Annee.id) as total_du
  FROM
    responsables AS Responsable,
    inscriptions AS Inscription,
    eleves AS Eleve,
    prestations AS Prestation,
    echeances AS Echeance,
    annees AS Annee
  WHERE
    Eleve.id=Inscription.eleve_id
    AND Responsable.id=Eleve.responsable_id
    AND Inscription.prestation_id=Prestation.id
    AND Inscription.annee_id=Annee.id
    AND Annee.id=Echeance.annee_id
    AND Inscription.id NOT IN (SELECT inscription_id
       FROM paiements)
    AND nombre_de_paiements>nb_echeances_finies_au(
       Echeance.id));
```

A.2 Extrait des *triggers*

```
-- ------------------------------------
-- Fonction qui cree les tarifs correspondants lors de
    l'affectation d'une prestation a une classe
-- ------------------------------------
CREATE TRIGGER
   apres_insertion_annees_classes_prestations AFTER
   INSERT ON annees_classes_prestations
FOR EACH ROW
BEGIN
  INSERT INTO tarifs(prestation_id,tranche_id,montant)
  SELECT DISTINCT
    NEW.prestation_id AS prestation_id,
    Tranche.id AS tranche_id,
    0.0 AS montant
  FROM
    tranches as Tranche
  WHERE
    Tranche.classe_id=NEW.classe_id
    AND Tranche.annee_id=NEW.annee_id;
END|
```

```
-- -------------------------------------
-- Fonction qui supprime les tarif lors de la
   desaffectation d'une prestation a une classe.
-- -------------------------------------
CREATE TRIGGER apres_suppression_classe_prestation
   AFTER DELETE ON annees_classes_prestations
FOR EACH ROW
BEGIN
  DELETE Tarif FROM
    tarifs AS Tarif,
    tranches AS Tranche
  WHERE
    Tranche.classe_id = OLD.classe_id
    AND Tranche.annee_id = OLD.annee_id
    AND Tarif.prestation_id = OLD.prestation_id
    AND Tarif.tranche_id = Tranche.id;

END|


-- -------------------------------------
-- Fonction qui supprime les tarif lors de la
   desaffectation d'une prestation a une classe.
-- -------------------------------------

CREATE TRIGGER apres_suppression_tranche AFTER DELETE
   ON tranches
FOR EACH ROW
BEGIN
DELETE IGNORE acp FROM
    annees_classes_prestations AS acp
  WHERE
  (acp.classe_id, acp.annee_id) NOT IN (SELECT
     DISTINCT classe_id,annee_id FROM tranches);

END|




-- -------------------------------------
```

```
-- Cree les lignes correpondantes dans la grille
   tarifaire lors de l'ajout d'un tranche a une classe
-- ----------------------------------

CREATE TRIGGER apres_insertion_tranche AFTER INSERT ON
    tranches
FOR EACH ROW
BEGIN
  INSERT INTO tarifs(prestation_id,tranche_id,montant)
  SELECT DISTINCT
    annees_classes_prestations.prestation_id AS
       prestation_id,
    NEW.id AS tranche_id,
    0.0 AS montant
  FROM
    annees_classes_prestations
  WHERE
    annees_classes_prestations.classe_id=NEW.classe_id
    AND annees_classes_prestations.annee_id=NEW.
       annee_id;
END|
```

A.3 Extrait des procédures stockées

```
SET GLOBAL log_bin_trust_function_creators = 1 |



-- ----------------------------
-- Retourne le tarif a l'annee qu'un eleve doit payer
   pour une annee donnee
-- ----------------------------
DROP FUNCTION IF EXISTS tarif_eleve_au|
CREATE FUNCTION tarif_eleve_au (eleve_id INT(11),
   anneeid INT(11)) RETURNS FLOAT(10,2)
BEGIN

  DECLARE quotient_responsable FLOAT(10,2);
  DECLARE prestation_eleve INT(11);
  DECLARE tranche INT(11);
  DECLARE annee INT(11);
  DECLARE classe INT(11);
```

```
DECLARE tarif FLOAT(10,2);

SET annee = anneeid;

SET quotient_responsable = (
                SELECT quotient_familial
                FROM responsables,eleves
                WHERE eleves.responsable_id =
                   responsables.id
                AND eleves.id=eleve_id LIMIT 1
              );

SET prestation_eleve = (
              SELECT prestation_id
              FROM eleves,inscriptions
              WHERE inscriptions.eleve_id = eleve_id
                AND annee_id=annee LIMIT 1
            );

SET classe = (
        SELECT classe_id
        FROM inscriptions
        WHERE inscriptions.eleve_id=eleve_id
          AND inscriptions.annee_id=annee
          LIMIT 1
      );

SET tranche =   (
          SELECT id
          FROM tranches
          WHERE quotient_min<=quotient_responsable
            AND classe_id=classe
            AND annee_id=annee
          ORDER BY quotient_min DESC
          LIMIT 1
        );

SET tarif =   (
          SELECT montant
          FROM tarifs
          WHERE tranche_id=tranche
            AND prestation_eleve=prestation_id
          LIMIT 1
```

```
          );

  RETURN tarif;
END |


-- ----------------------------
-- Retourne le montant du par un eleve a une echeance
   donnee
-- ----------------------------
DROP FUNCTION IF EXISTS du_encours_au|
CREATE FUNCTION du_encours_au(eleve_id INT(11),
   nombre_de_paiements INT(3),echeanceid INT(11),
   anneeid INT(11)) RETURNS FLOAT(10,2)
BEGIN
  DECLARE du_encours FLOAT(10,2);

  IF nb_echeances_finies_au(echeanceid) = 0 THEN
    SET du_encours = ROUND(tarif_eleve_au(eleve_id,
       anneeid)/nombre_de_paiements,2);
  ELSEIF nb_echeances_finies_au(echeanceid)+1>=
     nombre_de_paiements THEN
    SET du_encours = ROUND(tarif_eleve_au(eleve_id,
       anneeid),2);
  ELSE
    SET du_encours = ROUND((tarif_eleve_au(eleve_id,
       anneeid)/nombre_de_paiements)*(
       nb_echeances_finies_au(echeanceid)+1),2);
  END IF;

  SET du_encours = GREATEST(0,du_encours);


  RETURN du_encours;
END |



-- ----------------------------
-- Retourne le total percu pour une inscription et une
    echeance donnee.
-- ----------------------------
DROP FUNCTION IF EXISTS total_percu_echeance|
```

```
CREATE FUNCTION total_percu_echeance(inscription_id
   INT(11),echeance_id INT(11)) RETURNS FLOAT(10,2)
BEGIN
  DECLARE t FLOAT(10,2);
  SET t =     (SELECT
            SUM(montant)
          FROM
            paiements
          WHERE
            paiements.inscription_id=inscription_id
            AND paiements.echeance_id=echeance_id
          GROUP BY
            inscription_id,
            echeance_id);
  IF t IS NULL THEN
    RETURN 0;
  ELSE
    RETURN t;
  END IF;

END |

-- ----------------------------
-- Retourne le total percu pour une inscription donnee
-- ----------------------------
DROP FUNCTION IF EXISTS total_percu|
CREATE FUNCTION total_percu(inscription_id INT(11))
   RETURNS FLOAT(10,2)
BEGIN
  DECLARE t FLOAT(10,2);
  SET t =     (SELECT
            SUM(montant)
          FROM
            paiements
          WHERE
            paiements.inscription_id=inscription_id
          GROUP BY
            inscription_id
          );

  IF t IS NULL THEN
    RETURN 0;
  ELSE
```

```
    RETURN t;
  END IF;

END |


-- ----------------------------
-- Retourne le total percu pour une inscriptioon et
   une echeance donnee.
-- ----------------------------
DROP FUNCTION IF EXISTS total_percu_avant_echeance|
CREATE FUNCTION total_percu_avant_echeance(
   inscription_id INT(11),echeanceid INT(11)) RETURNS
   FLOAT(10,2)
BEGIN
  DECLARE t FLOAT(10,2);
  DECLARE datet DATE;

  SET datet =     (
                SELECT Echeance.date
                FROM
                  echeances AS Echeance
                WHERE
                  Echeance.id=echeanceid
              );



  SET t =     (SELECT
            SUM(montant)
          FROM
            paiements
          WHERE
            (paiements.inscription_id=inscription_id
               AND paiements.modedepaiement_id!=4
                AND paiements.echeance_id IN (SELECT
                   echeances.id FROM echeances WHERE
                   echeances.date<=datet))
            OR (paiements.inscription_id=
               inscription_id AND paiements.
               modedepaiement_id=4
            AND paiements.echeance_id IN (SELECT
               echeances.id FROM echeances WHERE
```

```
                echeances.date<datet))
            GROUP BY
              inscription_id);
  IF t IS NULL THEN
    RETURN 0;
  ELSE
    RETURN t;
  END IF;

END |


-- ----------------------------
-- Retourne le total percu pour une inscription donnee
   , sans les recouvrements du tresor public.
-- ----------------------------
DROP FUNCTION IF EXISTS total_percu_sans_recouvrements
   |
CREATE FUNCTION total_percu_sans_recouvrements(
   inscription_id INT(11)) RETURNS FLOAT(10,2)
BEGIN
  DECLARE t FLOAT(10,2);
  SET t =     (SELECT
              SUM(montant)
            FROM
              paiements
            WHERE
              paiements.inscription_id=inscription_id
              AND paiements.modedepaiement_id!=4
            GROUP BY
              inscription_id
            );

  IF t IS NULL THEN
    RETURN 0;
  ELSE
    RETURN t;
  END IF;

END |
```

```
-- ----------------------------
-- Retourne le nombre d'echeances finies pour l'annee
   en cours
-- ----------------------------
DROP FUNCTION IF EXISTS nb_echeances_finies |
CREATE FUNCTION nb_echeances_finies() RETURNS INT(11)
BEGIN
  DECLARE nombre_echeance INT(11);
  SET nombre_echeance =   (
                SELECT count(*)
                FROM
                  annees AS Annee,
                  echeances AS Echeance
                WHERE
                  Annee.etat='encours'
                  AND Echeance.etat='finie'
                  AND Annee.id = Echeance.annee_id
              );
  RETURN nombre_echeance;
END |


-- ----------------------------
-- Retourne le nombre d'echeances finies pour l'annee
   en cours a une echeance donnee
-- ----------------------------
DROP FUNCTION IF EXISTS nb_echeances_finies_au |
CREATE FUNCTION nb_echeances_finies_au(echeanceid INT
   (11)) RETURNS INT(11)
BEGIN
  DECLARE nombre_echeance INT(11);
  DECLARE datet DATE;
  DECLARE anneeid INT(11);

  SET datet =         (
                SELECT date
                FROM
                  echeances AS Echeance
                WHERE
                  Echeance.id=echeanceid
              );

  SET anneeid =       (
```

```
                SELECT annee_id
                FROM
                  echeances AS Echeance
                WHERE
                  Echeance.id=echeanceid
                LIMIT 1
              );


  SET nombre_echeance =   (
                SELECT count(*)
                FROM
                  echeances AS Echeance
                WHERE
                  Echeance.etat='finie'
                  AND Echeance.date<datet
                  AND anneeid = Echeance.annee_id
              );
  RETURN nombre_echeance;
END |




-- ----------------------------
-- Procedure qui ajoute un paiement recouvrement par
   le tresor public pour tous les impayes
-- ----------------------------
DROP PROCEDURE IF EXISTS recouvrement_automatique|
CREATE PROCEDURE recouvrement_automatique ()
BEGIN
  INSERT INTO paiements (inscription_id, echeance_id,
     modedepaiement_id, montant, date, emetteur,
     banque, numero_tresor_public)
    (SELECT
      inscription_id,
      ech.id as echeance_id,
      4 as modedepaiement_id,
      restant_du_encours as montant,
      CURRENT_DATE() as date,
      'Tresor public' as emetteur,
      NULL as banque,
```

```
        (SELECT MAX(numero_tresor_public)+1 as
           numtpsuivant FROM paiements) as
           numero_tresor_public
      FROM paiements_en_cours,
        (SELECT id FROM echeances WHERE echeances.etat='
           encours') as ech
      WHERE
        (inscription_id NOT IN (  SELECT inscription_id
                    FROM
                      paiements as Paiement,
                      echeances as Echeance
                    WHERE
                      Paiement.echeance_id=Echeance.id
                      AND Echeance.etat='encours'
                    )
        )
        AND restant_du_encours>0
      );

      INSERT INTO paiements (inscription_id, echeance_id
         , modedepaiement_id, montant, date, emetteur,
         banque, numero_tresor_public)
      (SELECT
        inscription_id,
        ech.id as echeance_id,
        4 as modedepaiement_id,
        restant_du as montant,
        CURRENT_DATE() as date,
        'Tresor public' as emetteur,
        NULL as banque,
        (SELECT MAX(numero_tresor_public) as
           numtpsuivant FROM paiements) as
           numero_tresor_public
      FROM paiements_en_cours,
        (SELECT id FROM echeances WHERE echeances.etat='
           encours') as ech
      WHERE
        (nb_echeances_finies()+1=nombre_de_paiements)
        AND restant_du>0
      );

  END |
```

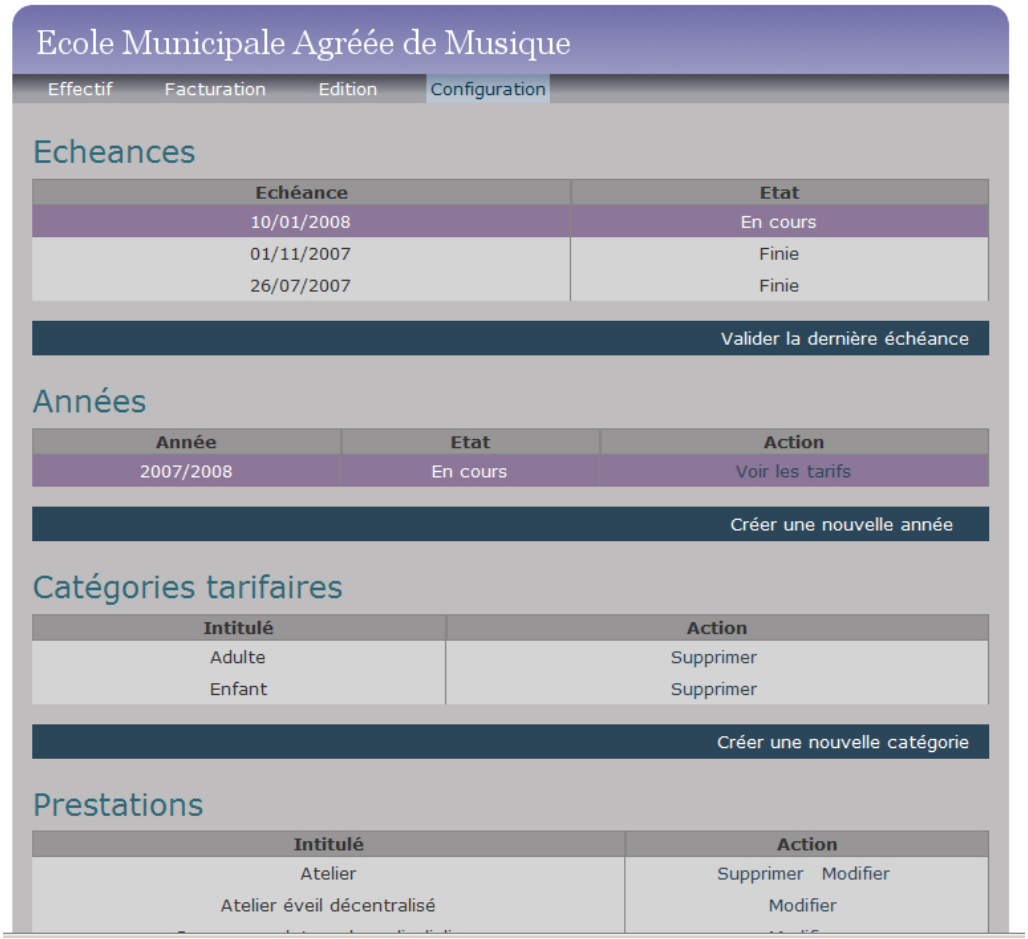
```
-- ----------------------------
-- Efface les tranches inutilises
-- ----------------------------
DROP PROCEDURE IF EXISTS purge_tranches|
CREATE PROCEDURE purge_tranches ()
BEGIN
DELETE Tranche FROM
    tranches AS Tranche
  WHERE
  (Tranche.classe_id, Tranche.annee_id) NOT IN (SELECT
      DISTINCT classe_id,annee_id FROM
     annees_classes_prestations);
END |


-- ----------------------------
-- Efface les associations annees classes prestation
   inutilisees
-- ----------------------------
DROP PROCEDURE IF EXISTS purge_acp|
CREATE PROCEDURE purge_acp ()
BEGIN
  DELETE IGNORE acp FROM
    annees_classes_prestations AS acp
  WHERE
  (acp.classe_id, acp.annee_id) NOT IN (SELECT
     DISTINCT classe_id,annee_id FROM tranches);
END |
```

A.4 Copies d'écrans

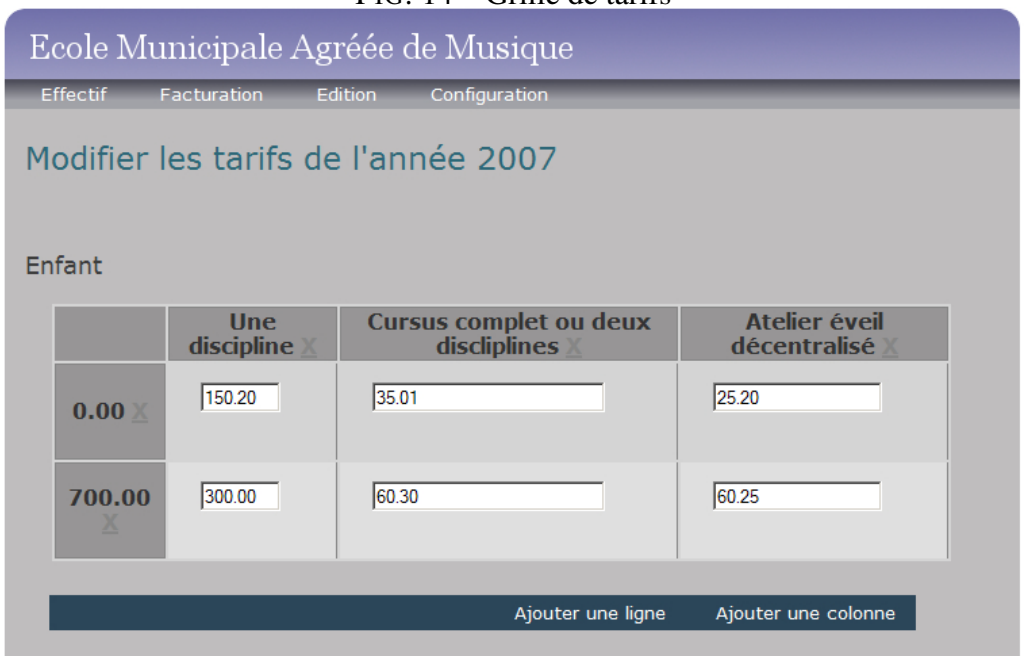
A.4.1 Page de configuration

FIG. 13 – Page de configuration des tarifs



A.4.2 Grille de tarifs

FIG. 14 – Grille de tarifs



A.4.3 Visualisation des inscriptions

FIG. 15 – Vue des inscriptions

A.5 Planning du stage

FIG. 16 – Planning du stage

Stage : Planning

ID	Tâche	Début	Fin	Durée	Dépend de
1	Familiarisation/Analyse de l'existant	02/04/07	06/04/07	4j	
2	Maquettage des interfaces	09/04/07	16/04/07	1sem	1
3	Modélisation des données	10/04/07	01/05/07	2sem 4j	1
4	Codage	18/04/07	29/05/07	5sem 4j	
5	Tests unitaires	01/05/07	08/06/07	5sem 3j	3
6	Tests utilisateurs	29/05/07	08/06/07	1sem 2j	4
7	Corrections	30/05/07	08/06/07	1sem 2j	

A.6 Facture générée par l'application

FIG. 17 – Facture générée par l'application

École de musique

Facture
Echéance du 10/01/2008

JULIEN TARTEMPION
22 AV DES OIES
12345 PERPETTES-LES-OLIVETTES

Pour Juliette TARTEMPION
Quotient familial : 1000.00
Inscrit en : Une discipline

Total de la facture : 60,20 €
A régler avant le 10/01/2008 à l'école de musique de Fontaine, par chèque à l'ordre du Trésor Public, en espèces, ou en chèques jeunes

Détail de la facture
Montant total des droits d'inscription pour l'année : 150,20 €
Montant dû au 26/07/2007 : 50,07 € Total des montants percus pour cette échéance : 10,00 € Solde à reporter : 40,07 €
Montant dû au 01/11/2007 : 50,06 + 40,07 = 90.13 € Total des montants percus pour cette échéance : 80,00 € Solde à reporter : 10,13 €

Ecole Municipale Agréée de Musique

22 rue des Alpes 38600 Fontaine - tel : 04.76.27.03.82 - Fax : 04.76.26.37.70 - ecole-musique@fontaine38.fr

B Glossaire

Glossaire

AGILE Une méthode AGILE est une méthode de développement informatique permettant de concevoir des logiciels en impliquant au maximum le demandeur (client), ce qui permet une grande réactivité à ses demandes. Les méthodes agiles se veulent plus pragmatiques que les méthodes traditionnelles. Elles visent la satisfaction réelle du besoin du client, et non d'un contrat établi préalablement. La notion de méthode agile est née à travers un manifeste signé par 17 personnalités (parmi lesquelles Ward Cunningham, l'inventeur du Wiki), créateurs de méthodes ou dirigeants de sociétés. (source : Wikipedia)

AJAX Asynchronous JavaScript and XML — Utilisation conjointe des technologies HTML/CSS/DOM/XMLHTTPREQUEST permettant d'établir une meilleure interaction utilisateur pour les pages WEB.

CSS Cascading Style Sheet — Langage de feuille de style adapté au page WEB.

CVS Concurrent versions system — Gestion concurrente des versions d'un code source.

Framework Ensemble de bibliothèque permettant le développement rapide d'applications complètes.

LDAP Lightweight Directory Access Protocol — Protocole permettant l'interrogation et la modification de services d'annuaire.

RAD Rapid Application Development — Méthode de développement de logiciel avec des cycles courts. Elle repose notamment sur l'utilisation d'outils qui permettent d'accélérer la production des interface graphiques, ce qui permet de créer des prototypes.

Smalltalk Smalltalk est un des premiers langages de programmation objet. Il a été créé en 1972. Il est inspiré par Lisp et Simula. Il a été conçu par Alan Kay, Dan Ingals, Ted Kaehler, Adele Goldberg au Palo Alto Research Center de Xerox. (source : Wikipedia).

SSO Single Sign On — Système d'authentification unique et standard.

Triggers Triggers ou Gachettes — En base de données, procédure stockée qui se lance dès la détection d'une action particulière sur la base.

XML eXtensible Markup Language — Méta-Langage de définition de formats de données.

C Bibliographie - Webographie

Références

[1] Équipe Analyse de l'IUT2
MERISE 2
2006.

[2] Dave SHEA
Le Zen des CSS
O'Reilly, 2005.

[3] Dave Thomas, David Heinemeier Hansson
Agile Web Development with Rails
The Pragmatic Programmers, 2005.

[4] *The Bakery, Everything CakePHP*
`http://bakery.cakephp.org/`

[5] *Article Ruby On Rails, Wikipédia*
`http://fr.wikipedia.org/wiki/Ruby_on_Rails`

[6] *CakePHP, site officiel*
`http://www.cakephp.org/`

[7] *Article Ergonomie, Wikipédia*
`http://fr.wikipedia.org/wiki/Ergonomie`

[8] *MODELS - VIEWS - CONTROLLERS, Trygve Reenskaug*
XEROX PARC 1978-79
`http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf`

[9] *phpMyAdmin*
`http://www.phpmyadmin.net`

[10] *SimpleTest PHP unit tester*
`http://www.simpletest.org/`

[11] Agnès CONTE and Mounia FREDJ, Jean-Pierre GIRAUDIN, Dominique RIEU
P-Sigma : un formalisme pour une représentation unifiée de patrons, 2001

D Table des figures

Table des figures

Résumé

Ce rapport traite de la réalisation d'un logiciel de gestion des effectifs et de facturation pour l'école de musique de Fontaine, durant un stage de deux mois au sein de la Direction des Systèmes d'Information (DSI) de cette ville.

Réalisée grâce à des technologies WEB, ce projet à permis d'appréhender l'utilisation d'un framework pour PHP orienté MVC, équipé de fonctionnalités RAD et de prototypage dans une démarche de développement itérative et centrée utilisateur.

Cette méthode de développement permise par les possibilités de ce framework vient avec une plus grande considération accordée à la qualité du logiciel, notamment l'utilisabilité.

Mots-Clefs

Fontaine — Facture — Élève — CakePHP — PHP — MySQL — Rapid Application Development — Prototypage

---

Summary

This document is about the building of invoices and pupils management software for the Fontaine's Music School, during a two months training course in the Information Systems Board of Fontaine town.

Built with web technologies, this project lets me take a tour at use of an MVC-oriented framework for PHP, with RAD features and prototyping capabilities, in an iterative and user centered way of project management.

This project management way allowed by this framework capabilities comes with more attention pay to some software quality criteria, usability for example.

Keywords Fontaine – Invoice – Pupil – CakePHP – PHP – MySQL – Rapid Application Development – Prototyping