Blue Ouest
Kerfiat
29720 PLONEOUR-LANVERN
Téléphone 02 98 87 14 52
Télécopie 02 98 66 00 35

Institut Universitaire de Technologie de LAVAL
Département Service et Réseaux de Communication
52 rue des Docteurs Calmette et Guérin
53000 LAVAL
Téléphone 02 43 59 49 20
Télécopie 02 43 59 49 28

Blue Ouest

Rapport de stage de fin de cursus

Réalisé par

LE GALL Nicolas

Du 20 mars 2006 Au 26 mai 2006

Soutenu le 2 juin 2006

Tuteur professionnel
Mme Elisabeth Quinquis

Tuteur enseignant
M. Yves Maussion

Je tiens à remercier Mme Quinquis, gérante de la société Blue Ouest et tuteur de mon stage, pour m'avoir accueilli dans son entreprise et pour m'avoir fait confiance en me confiant des missions importantes.

Je remercie M. Maussion, mon tuteur de stage et M. Schmidt, tous deux professeur à l'IUT de Laval, pour m'avoir suivi pendant mon stage.

SOMMAIRE

TABLE DES ILLUSTRATIONS

INTRODUCTION

J’ai effectué mon stage au sein de la société Blue Ouest, je présenterai les missions réalisées durant les deux mois de ce stage dans ce rapport.

Lors de mes démarches pour trouver une entreprise pour l’alternance j’avais pu faire la connaissance de Mme Quinquis de la société Blue Ouest. Depuis j’ai pu réaliser de petits projets pour elle. Effectuer mon stage dans son entreprise m’intéressait énormément car je pouvais effectuer des missions en choisissant ce que je voulais apprendre.

1. BLUE OUEST

La société Blue Ouest a été créée par Mme Quinquis en 2001 sous le statut d'entreprise individuelle. Implantée à son domicile, cela lui permet de ne pas avoir de frais de bureaux, et également de faire n'importe quels horaires (8h – 22h le plus souvent).

La société propose des services d'édition, de formation à domicile, de réalisation de CDROM ainsi que de sites web. Mais n'employant pas de programmeur, elle est obligée de sous-traiter. Cela est un problème, car les programmeurs indépendants qui font ces travaux n'ont pas vraiment d'horaires fixes, et certains projets ont du avorter à cause de cela. Certains projets ont même dû être refusés, le travail demandé étant trop conséquent et la masse de travail en retard considérable.

En 2004 Blue Ouest lance le site « Chronium », un agenda numérique développé par une société de Brest. Depuis la société fait essentiellement de l'édition, des projets de sites web sont en préparation mais nécessitent un programmeur.

2. LES MISSIONS

2.1 Gestionnaire de nouvelles

2.11 Analyse des besoins

Blue Ouest, n'employant pas de programmeur, est obligé de sous-traiter tous les travaux nécessitant de la programmation. La plupart des sites réalisés ont besoin d'un module de gestion de nouvelles. C'est dans cette optique que ma première mission a été de développer un module capable de s'adapter à n'importe quel site et ce très rapidement.

Le module ne devait pas faire intervenir de PHP lors de son intégration dans un site, et également le minimum de HTML. Les possibilités de personnalisation devaient être maximales, et la mise en page ne devait pas dépendre du module et se faire très simplement également. Les adresses générées par le script devaient être simples et tenir compte des autres scripts pouvant s'exécuter sur le site, et ainsi gérer sa dépendance automatiquement.

L'édition ou la création d'une nouvelle devait être très simplifiée, mais permettre à l'utilisateur de personnaliser son texte de la même manière qu'un traitement de texte. Un gestionnaire d'image devait faire partie intégrante du module, permettant de rajouter des images pour illustrer une nouvelle. Lors de l'ajout d'une image (upload) l'utilisateur devait pouvoir continuer à éditer son texte.

De plus, le module devait être suffisamment sécurisé pour ne pas être une faille pour le site. Il devait générer du HTML propre et respectant les standards, ce qui permet une meilleure personnalisation. Le code PHP devait être simple et facilement modifiable par un autre programmeur, pour l'ajout de nouvelles fonctionnalités par exemple.

2.12 Analyse des contraintes techniques

Avant tout, le module devait être simple à personnaliser par l'intégrateur, pour lui permettre d'adapter le rendu avec le site. Pour ce faire il fallait utiliser un moteur de template. Or un moteur de template à des défauts :

- Le script est souvent lourd et ralenti l'affichage du site.
- Aucun moteur n'utilise les mêmes codes et conventions, leurs utilisations nécessitent donc un temps d'adaptation.
- Le module étant relativement petit, utiliser un moteur de template aurait été démesuré.

Tous ces défauts m'ont amené à développer un mini moteur de template, uniquement adapté à ce module.

Le module ce devait d'être dépendant du site, c'est-à-dire de fonctionner seul mais de ne pas empêcher d'autres scripts ou le site de fonctionner. Cette contrainte agit sur deux niveaux :

- Il faut surveiller l'adresse pour garder les variables qu'un autre script peut utiliser.
- Le script devait savoir ou il se situait dans la structure de fichier (la structure dépendant du système de fichier du serveur qui diffère d'Unix à Windows).

L'espace d'administration permettant de créer les articles se devait d'avoir une « boite à outils » pour que l'utilisateur puisse personnaliser son texte. Cette boite à outils devait donc agir sur le navigateur de l'utilisateur, et ce grâce au langage JavaScript. C'est une contrainte car le langage diffère légèrement d'un navigateur à un autre (principalement d'Internet Explorer à Firefox / Opera / Safari), il a donc fallu adapter le code pour qu'il soit interprétable pour l'ensemble des navigateurs.

L'utilisateur n'étant pas censé connaître le HTML, l'édition d'un article se devait d'utiliser des « codes » signalant au script que l'utilisateur souhaite personnaliser une partie de texte. Ce mode d'édition est appelé Wiki. Il m'a fallu définir les caractères que l'utilisateur pourrait insérer pour personnaliser manuellement son texte (sans la barre d'outils, mais un click sur une icône de celle-ci fait la même chose). Il s'agit d'une contrainte car il y a beaucoup de traitement à faire lors de l'enregistrement de l'article, de plus il a fallu trouver des caractères facilement accessibles, et susceptibles de ne pas être utilisés dans le texte.

Le module devait fonctionner avec une version de PHP et de MySQL (le support d'un autre SGBD n'était pas prévu) la plus ancienne possible, pour être exploitable sur n'importe quel serveur.

2.13 Analyse concurrentielle

Ce module s'apparente plus à un système de blog qu'a un CMS ou un Wiki. J'ai donc réalisé une rapide analyse concurrentielle des systèmes de blogs les plus répandus.

- DotClear : DotClear est un logiciel libre de blog. Il est très apprécié de la communauté du logiciel libre français. Ces principaux avantages sont :
    - Un code conforme aux normes du W3C et un usage très présent du CSS.
    - Une personnalisation sans limite.

- Une syntaxe d'édition en Wiki et xHTML.
- Fonctionne sur la majorité des hébergeurs gratuits ou non.
- Une bonne sécurité.

Mais utiliser ce système pour chaque site est trop démesuré. DotClear est très bien pour un usage à part, en tant que blog et rien d'autre (ou en tant que site, mais pas en tant que module pour un site). De plus son interface d'administration n'est pas personnalisable, ce qui provoquerait une incohérence sur le site.

- WordPress : WordPress est le principal concurrent de DotClear, mais il remporte un plus grand succès à l'extérieur de l'Hexagone. Ces avantages sont sensiblement les même que DotClear. Et ici aussi il est impossible de dissocier l'interface d'administration.

L'utilisation d'un système de blog était donc à proscrire, les fonctionnalités sont bonnes, mais l'ensemble est trop lourd, et non adapté à un fonctionnement modulaire.

2.14 Organisation, architecture

J'ai décidé d'organiser tout de suite le système de fichier pour qu'il soit le plus simple possible. Voici la structure pour le module qui affiche les nouvelles (du côté public donc) :

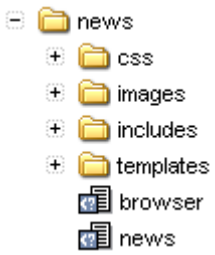


Figure 1. Structure des dossiers pour le module d'affichage

- « news » : contient tout le module, ce nom de dossier est laissé à la liberté de l'intégrateur.
- « css » : ce dossier est facultatif (les styles CSS pouvant être déclarés dans la feuille de style du site), ici il contient les feuilles de style utilisés par le module, ainsi que les images.
- « images » : sert à accueillir les images que l'utilisateur a téléchargé.
- « includes » : contient les fichiers nécessaires aux scripts comme le fichier de configuration et les fonctions.

- « templates » : contient les fichiers de templates permettant à l'intégrateur de personnaliser le HTML généré par le module.
- « browser.php » : Il s'agit du script qui permet la navigation dans le site, pour consulter les nouvelles antérieures (un fichier de configuration renseigne sur le nombre de nouvelles à afficher).
- « news.php » : Le module en lui-même, il affiche un article ou une liste d'articles.

Voici la structure pour le module de gestion (administration) :

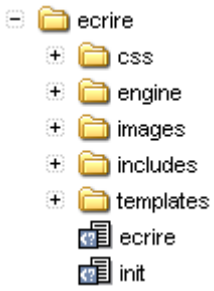


Figure 2. Structure des dossiers pour le module d'administration

- « ecrire » : contient tout le module, ce nom de dossier est laissé à la liberté de l'intégrateur.
- « css, includes et templates» : identique que le module d'affichage.
- « engine » : Le moteur d'administration, permet de lister les articles déjà publiés ainsi que la page permettant d'éditer / créer un article.
- « images » : Contient les images de la boite à outils (les icônes permettant à l'utilisateur de mettre en forme son texte).
- « ecrire.php » : Le module qui charge le moteur d'administration.
- « init.php » : Un script indispensable à l'initialisation du moteur (doit être chargé en premier dans la page).

L'organisation du travail était fastidieuse à établir, il s'agit d'un module qui ne fonctionne que si chaque élément est opérationnel. J'ai d'abord commencé par créer la base de données (BDD), j'y ai ajouté 3 articles basiques avec du texte de substitution. A partir de cette base j'ai pu commencer par créer le module d'affichage, qui listait les articles de la BDD (en tenant compte de la limite d'article par page que j'avais délibérément fixé à 2). J'ai ensuite pu m'attaquer au mini moteur de template, ce moteur devait utiliser les fichiers de template et remplacer les variables par

le contenu de la BDD (plus d'explication dans la sous partie 2.16). Le moteur devait répéter l'opération pour chaque article. Le module d'affichage était presque complet, il ne me restait qu'à créer le module de navigation (en utilisant le même principe).

J'ai ensuite commencé le module d'administration en calquant le fonctionnement de l'affichage. Il ne me restait plus que l'éditeur Wiki à faire ainsi que la boite à outils.

2.15 La base de données

La base de données était très simple et très petite. Le module ne gérant que le strict minimum d'informations :

- Le titre de l'article
- Un sous-titre (facultatif)
- La date de publication
- Le texte
- L'état de l'article (en ligne / hors ligne)
- Un identifiant (obligatoire pour gérer les données sur le serveur)

Le contenu de la base étant maigre il n'était pas nécessaire d'optimiser les tables.

2.16 Conventions, codage, technologies utilisées

Avant de commencer à programmer le module il m'a fallu définir les conventions que j'allais appliquer.

Pour les templates :

- Une extension de fichier .tpl (abréviation commune de « template »).
- Un nom de fichier en anglais, une segmentation maximale des fichiers pour permettre une bonne personnalisation.
- Des variables encadrées d'accolades et en anglais (pour afficher le titre par exemple : {title})

  J'ai opté pour des accolades, certains moteurs de templates utilisent les signes de balisage < et >, mais je n'ai pas souhaité les utiliser. En effet, ces signes sont utilisés dans les langages de balisage comme le HTML, xHTML ou encore XML. Il me semble trop

confus d'utiliser cette méthode pour définir un contenu sémantique différent (les balises indiquent ce que le contenu représente, et non le contenu lui-même).

- Une distinction sémantique de la variable : {title} pour afficher le titre, {link:title} pour créer un lien sur le titre.

Pour le PHP :

- Une structure identique pour tous les fichiers :
    - Chargement des fichiers nécessaires.
    - Initialisation et vérification des variables.
    - Connexion à la BDD, récupération des données, déconnexion de la BDD (monopolise moins le serveur).
    - Récupération des templates.
    - Traitement des templates et des données.
    - Affichage.

Cette structure est très efficace car très ordonnée, les données venant de l'utilisateur sont vérifiées en premier ce qui augmente la sécurité, l'affichage en dernier permet d'avoir de meilleures performances. J'utilise cette structure depuis mon stage court.

- Des variables en anglais, le plus explicite possible (permettent une meilleure maintenance, et une meilleure portabilité).
- Des fichiers courts et espacés, une indentation correcte et des lignes courtes (maximum : 200 lignes, 150 caractères par ligne).

Pour le JavaScript :

- Des variables en anglais, des fichiers courts (identique au PHP).
- L'utilisation du DOM.

Pour le Wiki (édition d'un article) :

- __texte__ : met le texte en gras.
- ''texte'' : met le texte en italique.
- ++texte++ : souligne le texte.
- --texte-- : barre le texte.

- ||texte|| : crée un bloc de citation (utilisé pour mettre en forme, par exemple une bulle indiquant qu'il s'agit d'une citation).
- &&texte&& : insère le texte en « pré formaté » (tel qu'il a été tapé, en respectant les tabulations, et les sauts de ligne, permet également d'être mis en forme différemment).
- {{texte}} : crée une citation en ligne (moins de possibilité de mise en forme que le bloc).
- [lien|adresse|langue] : crée un lien hypertexte, « lien » est le texte affiché et cliquable, « adresse » est l'adresse URL de destination, « langue » est facultatif et permet de renseigner l'utilisateur sur la langue du lien qu'il va visiter (cette fonctionnalité doit être activée grâce au CSS).
- ((image.jpg|texte)) : crée une image en utilisant « image.jpg », « texte » est facultatif et représente le texte alternatif à afficher si l'image ne peut être affichée (si l'image n'existe pas, ou pour les lecteurs d'écrans).
- Les sauts de lignes sont traduit en HTML (un saut de ligne est représenté par le code ASCII '\n', hors en HTML un saut de ligne est réalisé grâce à la balise
).
- Les listes sont supportées mais ont été désactivées, le développement de la fonction de conversion prenant trop de temps et avait un fonctionnement trop hasardeux (pour information, les listes étaient représenté par une étoile (*), suivi du texte et d'un saut de ligne ; ensuite une autre étoile créait une nouvelle puce ; le problème vient du fait que les listes font intervenir deux balises en xHTML).
- Bien entendu, l'utilisateur peut combiner ces « codes », il peut faire un lien sur une image par exemple : [((image.jpg|texte))|adresse|langue] ; ou encore un texte en gras et souligné : ++__texte__++ (l'ordre n'a pas d'importance, __++texte++__ donne le même résultat).
- L'utilisateur peut spécifier qu'il souhaite afficher les caractères plutôt que d'appliquer la mise en forme. Il lui suffit d'insérer le caractère backslash (\ ) avant le mot. Par exemple : « __texte__ » affiche « __texte__ » et non « **texte** ».

L'utilisation de variables en anglais permet souvent d'avoir des noms plus petits donc qui consomment sensiblement moins de ressources, elles peuvent être plus facilement mémorisé, gardent une certaine cohérence avec le langage (PHP utilise des noms de fonctions et de variables en anglais, comme la plupart des langages de programmation) et dissocient plus facilement le programme du rendu (ou contenu).

Une fois les conventions définies j'ai pu commencer le code. Je me suis d'abord focalisé sur le mini moteur de template. Son fonctionnement est très simple, il s'agit d'un script qui ouvre un

fichier de template, et qui remplace les variables qu'il trouve ({title} par exemple) par le contenu à afficher.

Pour développer ce script j'ai du utiliser les expressions régulières que je n'avais jamais utilisé auparavant (le sujet est assez rebutant de part la complexité des expressions). Sous ce nom énigmatique, se cache en fait un ensemble de fonctions simples : trouver / remplacer du texte , et ce en utilisant une expression. Par exemple (en simplifiant) :

```
[a-zA-Z]@[0-9] : demande de trouver une expression tel que « une lettre majuscule ou non, suivie d'un @ puis d'un chiffre ».
```

Les possibilités offertes sont infinis et plus l'expression à rechercher est complexe plus l'expression régulière le sera. Pour l'appliquer au moteur de template il suffisait de spécifier les variables de template en expression régulière et de demander à la fonction de la remplacer par le contenu correspondant récupéré dans la BDD.

L'application commençait à fonctionner (du moins les templates, le module de création d'article n'étant pas encore développé), je me suis ensuite penché sur l'autonomie du script. Il n'était pas question de renseigner au script son répertoire d'installation dans le fichier de configuration, les chemins diffèrent d'un système d'exploitation à un autre (pour Unix de la forme : « home/var/www/html » ; pour Windows : « c:/www/html ») de plus on ne connaît que très rarement le chemin exact du fichier sur le serveur (pour une raison de sécurité). Il a fallu développer une fonction qui trouve où le module est installé par rapport à la page qui le charge. Cette fonction ne peut malheureusement pas fonctionner dans 100% des cas, trop de paramètres rentrant en compte.

J'ai ensuite terminé l'application en créant la boite à outils ainsi que la fonction de conversion Wiki. Cette dernière fonctionne uniquement grâce à des expressions régulières plus ou moins complexes :

```
#(?<!\\\\)\[([^|]*?)\|([^|^\]]+)(\|(([^\]]*?)+))?\]# : cette expression permet de vérifier qu'un lien a été tapé dans l'article (voir les conventions du Wiki).

#^http://[[:alnum:]]*\.*[[:alnum:]]{2,}\.[[:alpha:]]{2,4}$# : vérifie la validité d'une adresse Internet (« http:// » suivi de plusieurs ou aucun caractères, d'un point ou non, de 2 caractères minimum, d'un point obligatoire et de deux, trois ou quatre caractères alphabétiques).
```

Comme précisé précédemment, la boite à outils devait être en JavaScript, ce langage permettant d'agir sur le navigateur de l'utilisateur (une capture d'écran est disponible en annexe). Ce script devait insérer les caractères définis dans les conventions Wiki, ou en encadrer le texte

sélectionné. La principale difficulté était d'agir sur la zone de texte, Internet Explorer (IE) gère différemment les fonctions que Firefox ou Opera.

```
Pour récupérer le texte sélectionné sous IE :

var selection = document.selection.createRange();

L'équivalent pour les autres navigateurs :

var start = textarea.selectionStart;
var end = textarea.selectionEnd;
var selection = textarea.value.substring(start, end);
```

Le gestionnaire d'image n'était pas prévu au départ, mais le module était terminé plus tôt que prévu et ce genre de gestionnaire est relativement simple à développer. Il a donc été rajouté en fin de mission, son utilisation est très simple : lors de l'édition de l'article, la boite à outils propose une icône d'insertion d'image. En cliquant dessus on ouvre une nouvelle fenêtre listant les images déjà présentes sur le serveur, si l'image que l'utilisateur souhaite insérer ne s'y trouve pas un simple bouton « parcourir » permet de la rajouter. En cliquant ensuite sur le lien correspondant on insère le « code » dans la zone d'édition d'article (en annexe une capture d'écran du gestionnaire). Le codage n'a pas posé de difficulté et a pris un jour et demi.

2.17 Problèmes persistants

Il reste toutefois des bugs dans ce module, ou plutôt le module peut créer des bugs lors de son intégration dans un site :

- Il peut causer un problème avec les connexions à la BDD. Il ne s'agit pas d'un problème du module en lui-même, mais de la fonction PHP qui réalise les connexions et les requêtes. Il est spécifié dans la documentation PHP (à propos de la fonction qui effectue les requêtes): « *La connexion MySQL. S'il n'est pas spécifié, la dernière connexion ouverte avec la fonction* ***mysql_connect()*** *sera utilisée. Si une telle connexion n'est pas trouvée, la fonction tentera d'ouvrir une connexion, comme si la fonction* ***mysql_connect()*** *avait été appelée sans argument.* »

  Le modulé développé spécifie systématiquement la connexion à utiliser, mais il se peut que le reste des scripts du site ne le fasse pas. PHP essayera d'utiliser la connexion du module alors qu'elle est fermée automatiquement, ce qui produira une erreur. Ce bug n'est pas simple à corriger, le meilleur moyen aurait été d'utiliser un objet pour se connecter à la BDD, mais la 2ème mission devait me permettre d'appréhender la programmation orientée objet.

- Il se peut que le site utilise des noms de fonctions et de variables identiques au module, ce qui créera un conflit. Une fois de plus, ce n'est pas un bug simple à corriger, la programmation orientée objet aurait grandement solutionné tous ces problèmes, mais il s'agit de bugs rencontrés à la fin de la mission et d'importance moindre.

2.18 Manuel d'utilisation / référence

J'ai également réalisé un manuel d'utilisation pour le module, permettant à l'intégrateur d'accéder rapidement à la documentation. Ce manuel couvre les points suivants :

- Configuration minimum requise.
- Installation manuelle (une installation automatique était initialement prévue, mais l'installation manuelle étant très simple il n'a pas été indispensable de la développer).
- Les variables de templates : liste les variables disponibles ({title}, {text}, etc…)
- Les sélecteurs CSS : liste les sélecteurs CSS imposés, le côté public du module n'en a pas.
- Exemple (côté public) : un exemple de template et de CSS pour le côté public.
- Exemple (administration) : un exemple de template et de CSS pour l'administration.
- Les variables en méthode GET utilisées : informe des variables que le module utilise, pour adapter les autres scripts en cas de conflits.
- Bugs connus : Une explication des bugs connus (voir sous section précédente).

Vous pouvez consulter ce manuel à l'adresse suivante : http://www.neovov.com/blue/news/manual (l'utilisation d'Internet Explorer est fortement déconseillée, ne supportant pas des fonctions xHTML/CSS utilisées).

2.2 Gestionnaire de sondage

2.21 Analyse des besoins

Blue Ouest prépare actuellement un portail d'information sur le pays bigouden. Le site devant être proche des gens il était naturel d'avoir l'avis des internautes sur les sections qu'ils aimeraient voir apparaître. C'est pourquoi il était nécessaire de faire un module de sondage.

Les fonctionnalités devaient rester simples ; la possibilité de poser une question, de définir les réponses possibles, et de mettre en ligne une question. Il devait bien sûr être possible de supprimer

un sondage. Un sondage est terminé lorsqu'il n'est plus en ligne, aucune limite de temps n'est imposée ou définie. Une fois de plus l'intégration devait être simple, le module pouvait être rajouté sur d'autres sites.

Du côté public l'utilisateur ne devait pouvoir répondre qu'une seule fois au sondage en cours. Une fois le vote effectué les résultats devaient s'afficher à la place des réponses (sous forme de graphiques).

J'ai choisi d'effectuer ce module en utilisant deux technologies que je voulais essayer depuis longtemps : le PHP orienté objet et l'Ajax. Cette mission me semblait parfaitement adaptée pour apprendre correctement les bases de chacune de ces techniques.

2.22 Analyse des contraintes techniques

L'administrateur devait pouvoir créer les sondages à la volé, c'est-à-dire sans les avoir préparé au préalable sur papier. Avec un usage traditionnel (PHP & xHTML) à chaque nouvelle action l'utilisateur est obligé d'attendre le chargement de la nouvelle page pour continuer (par exemple pour rajouter une question si au dernier moment il juge qu'il faut en rajouter une). C'est en analysant cette contrainte que j'ai décidé de faire une page entièrement dynamique, c'est-à-dire sans chargement entre chaque action. Ceci est possible grâce au JavaScript, à la manipulation du DOM et à l'Ajax qu'il m'a fallu étudier au préalable.

J'ai imposé une contrainte technique pour l'ensemble du site également : j'ai vivement conseillé l'utilisation de PHP5. Le portail va principalement utiliser des flux RSS pour enrichir son contenu, et PHP5 intègre des fonctions de gestion XML très avancées en natif (il aurait été très fastidieux de recoder les fonctions intégré à PHP5 en utilisant PHP4).

2.23 Organisation, études préalables

J'ai commencé par scinder le travail en deux parties : le PHP et le JavaScript. Le PHP allait permettre d'effectuer tous les traitements nécessaires, et le JavaScript à faire une page entièrement dynamique.

Pour m'organiser dans mon travail de développement PHP j'ai d'abord commencé par faire une analyse des classes qu'il me fallait, puis je les ai modélisées en UML grâce au logiciel WaterProof::UML. Ce logiciel permet d'avoir un rendu graphique des classes qui vont être codées, et ainsi de voir les relations d'héritage.

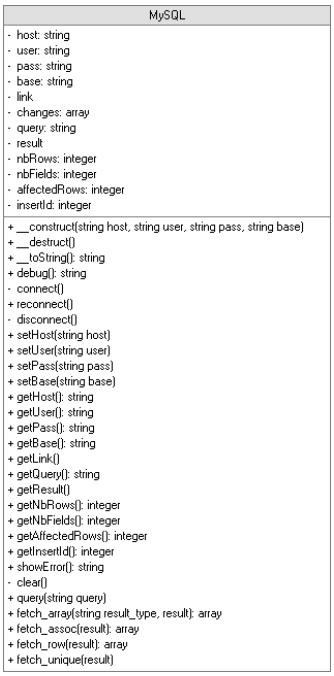


Figure 3. Ma classe MySQL modélisée en UML (voir annexe)

Il permet également de transformer le rendu graphique en code, en déclarant les prototypes des méthodes qu'il ne reste plus qu'à coder. J'ai choisi de coder mes classes en PHP5, pour se rapprocher au maximum de la programmation orienté objet tel que JAVA ou C++, et ce en déclarant les portées des variables et méthodes. Un des gros avantage de la programmation orienté objet est la portabilité du code, il suffit de charger une classe pour s'en servir (on évite ainsi de recoder des fonctions que l'on a déjà fait, et on évite le copier-coller).

Il m'a fallu étudier l'Ajax, et plus particulièrement l'objet XMLHttpRequest, qui en est la base même (voir annexe pour une explication plus approfondie). Et surtout le mécanisme des transactions effectuées en utilisant l'objet. En effet, il permet de contacter un fichier présent sur le serveur, de lui envoyer des données et d'attendre la réponse. J'ai donc du faire une structure pour les dialogues JavaScript – PHP :

- Traitements, vérifications des données à envoyer.
- Affichage d'un indicateur d'activité (informe l'utilisateur que le script travaille en arrière plan).
- Envoi des données sur le script PHP.
- Exécution d'une fonction spécifique lorsque le script PHP à répondu.
- Suppression de l'indicateur.

2.24 La base de données

Cette mission m'a permis d'approfondir ma connaissance dans les bases de données (BDD), et plus particulièrement dans les bases relationnelles. Ce genre de base permet de ne pas avoir à se soucier de données orphelines quand plusieurs tables se complètent ou sont liés d'une certaine façon. Dans le cas d'une BDD non relationnelle, c'est le script qui agit sur les données qui doit s'occuper des données orphelines. Par exemple : prenons deux tables, « questions » et « réponses ». La table questions est composée d'un identifiant et de la question ; la table réponses quand à elle est composée d'un identifiant, de l'identifiant de la question à laquelle elle est associée, et de la réponse.

- Avec des tables non relationnelles : si on supprime une question, les réponses qui y sont associées ne sont pas supprimées. On se retrouve donc avec des données orphelines, sans raison d'exister.
- Avec des tables relationnelles : si on supprime une question, les réponses associées sont automatiquement supprimées.

On peut obtenir des effets en cascade très intéressants, et qui permettent de gagner du temps lors de la programmation. Ces relations sont possibles grâces aux clés étrangères (définies grâce à la méthode MERISE). En utilisant un programme de modélisation de BDD on peut aisément conceptualiser les tables :

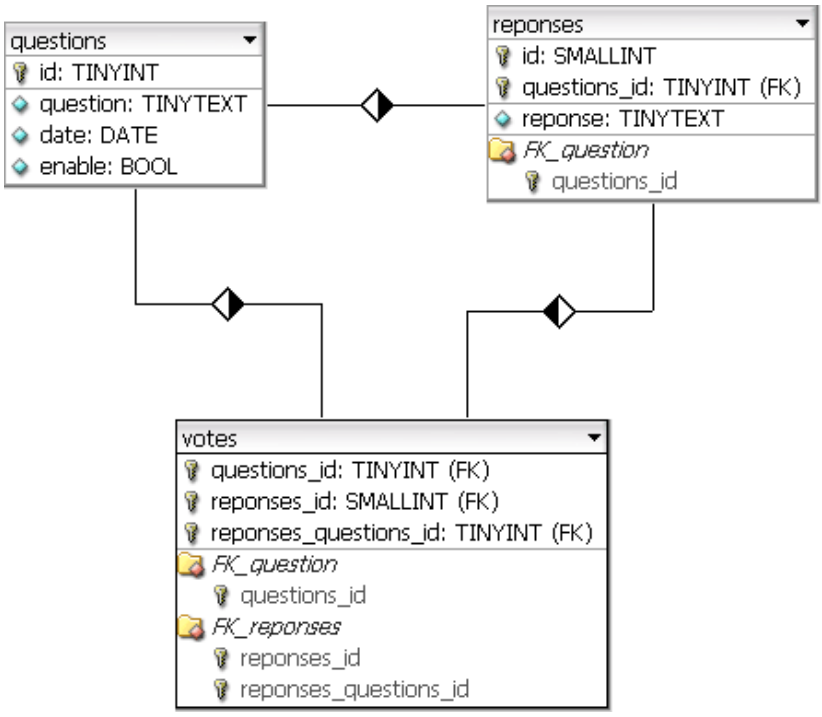


Figure 4. Modélisation de la base de données

Voici la structure finale de la BDD :

- Table questions :
  - id : l'identifiant de la question.
  - question : la question.
  - date : la date de création de la question.
  - enable : le statut (en ligne / hors ligne).
- Table réponses :
  - id : l'identifiant de la réponse.
  - questions_id : l'identifiant de la question qui concerne la réponse.
  - reponse : la réponse.
- Table votes :
  - questions_id : l'identifiant de la question.
  - reponses_id : l'identifiant de la question.
  - reponses_questions_id : Il s'agit d'un champ parasite généré par le programme, ce champ ne fait pas parti de la BDD finale.

2.25 Codage

Après la modélisation des classes en UML, le codage PHP à été très rapide. En effet le programme converti le modèle en prototypes, il ne reste plus qu'a définir le code à mettre dans les méthodes. Une longue période de test à quand même été nécessaire pour vérifier que les classes fonctionnaient bien, et qu'il ne manquait aucune méthode. La classe la plus important a été celle de connexion à la BDD, j'ai particulièrement accentué le coté débogage, ce qui permet d'avancer plus vite lorsque l'on rencontre une erreur (PHP ne renseigne pas toujours parfaitement lorsqu'une erreur se produit).

Il m'a fallu ensuite trouver un moyen pour que les objets puissent accéder à la base de données. En effet un objet une fois créé ne connaît rien de ce qui l'entoure, il faut lui indiquer ce qu'il doit utiliser. J'ai donc utilisé ma classe MySQL pour cela, et la souplesse de l'objet (et surtout du PHP5) m'a permis rapidement de mettre en place un « *connecteur* », pour que chacune des classes aient accès aux données et également pour qu'elles puissent interagir avec.

```
$question = new Question('Pourquoi ?');
$question->save($sql);
```

Cet exemple très simple montre la façon de créer une nouvelle question, et sa sauvegarde.

On crée d'abord l'objet en lui donnant les paramètres adéquats (ici l'intitulé de la question), et on stocke l'objet dans la variable '*$question*'. Pour sauvegarder on appel la méthode '*save*' de l'objet *$question* (on demande à *$question* de se sauvegarder), en lui fournissant le « *connecteur* » (l'objet utilisé pour interagir avec la BDD).

Cette souplesse de programmation permet de travailler plus vite, d'avoir du code plus lisible, plus proche du langage naturel, et aussi de gérer des opérations plus complexes (ou qui auraient été plus complexe en programmation traditionnelle).

Il m'a fallut ensuite faire des scripts pour l'interaction avec Ajax. J'ai décidé de nommer ces scripts des « *bridges* » (passerelles), car il s'agit de scripts de transition permettant à JavaScript d'obtenir du contenu ou de le modifier. Ces scripts sont relativement simples car il y a peu de traitements à faire, et la structure est assez proche du langage naturel. Le codage de ces « *bridges* » à du se faire à la fin du travail de JavaScript.

```
if(preg_match('#new_question#', $id)){
$question = new Question($text);
$question->save($sql);
}
```

Littéralement : Si *$id* contient « new_question », on crée une question avec le *$text* fourni, puis on sauvegarde.

La partie JavaScript était également très intéressante. J'ai d'abord commencé par étudier l'objet XMLHttpRequest, et j'ai réalisé une classe pour m'en servir le plus simplement possible. En effet pour maximiser la compatibilité il est nécessaire d'effectuer des vérifications avant de pouvoir ouvrir la connexion (par exemple : avec Internet Explorer il faut utiliser l'objet *ActiveXObject("Msxml2.XMLHTTP.5.0")* alors qu'avec le reste des navigateurs il faut utiliser *XMLHttpRequest*()). La construction de cette classe n'a pas pu se faire grâce à un logiciel, JavaScript étant un langage qui souffre d'une faible documentation et de très peu de logiciels d'aide au développement.

```
xhr = new XHRConnexion();
xhr.appendData('id', id_enCours);
xhr.appendData('text', text);
xhr.send('POST', 'bridge/save.php', saveCallback);
```

Ceci illustre l'utilisation de ma classe, on commence par créer l'objet, puis on utilise une méthode pour rajouter des données à transmettre dans une pile. Ensuite on envoi les données avec la méthode « *send* », sur le bridge servant à sauvegarder. Une fois que le bridge à répondu, on exécute la fonction « saveCallback ».

Pour utiliser l'Ajax il faut particulièrement bien connaître le DOM. Hors JavaScript manque cruellement de documentation, il m'a donc fallu tâtonner pour réussir.

```
var indicator = document.createElement('div');
var indicatorText = document.createTextNode(' Connexion...');
var indicatorImg = document.createElement('img');
indicator.appendChild(img);
indicator.appendChild(indicatorText);
document.getElementById("page").appendChild(indicator);
```

Cet exemple illustre l'utilisation du DOM pour rajouter un indicateur d'activité. En fait il s'agit juste de création de HTML (le code si dessus est une version allégée).

Comme dit précédemment, il n'existe aucun logiciel de modélisation de classe pour JavaScript, la mise en œuvre d'une application comme celle-ci était donc relativement plus longue. Le code final n'est pas très long (~ 300 lignes), mais la difficulté était de maintenir une compatibilité entre chaque navigateur, et ce malgré l'utilisation du DOM.

2.26 Problèmes persistants

Il n'y a pas de problèmes persistants, toutefois la gestion du JavaScript est différente d'un navigateur à un autre, il peut donc y avoir des problèmes sur les navigateurs non testés.

Le module à été conçu pour fonctionner avec JavaScript, les navigateurs qui l'ont désactivé ou qui ne le supportent pas ne permettront pas d'utiliser le module. Mais ce genre de navigateur est assez rare, de plus l'administrateur qui doit gérer le site est prévenu si le JavaScript est désactivé.

2.3 Moteur de recherche

2.31 Analyse des besoins

Toujours dans le cadre du portail sur le pays bigouden, j'ai eu à ma charge la création de la base de données dans son ensemble. Le portail devra fournir une liste d'entreprises implantées dans le pays bigouden, ainsi qu'un maximum d'informations les concernant. Cette dernière

mission consistait à travailler uniquement sur la BDD, de la création en passant par son remplissage et un moteur de recherche également.

Cette BDD se devait d'être le plus souple possible, c'est-à-dire une évolution le moins compliquée possible. Les données devaient être structurées de la manière la plus optimisée possible, la base devant contenir une quantité relativement importante d'informations (~ 2000 entreprises par exemple). Le remplissage de la BDD devait se faire simplement, rapidement et automatiquement. Les données de bases étant saisies dans Excel, ou récupérées grâce à un programme de type « aspiration de données ».

L'utilisateur devait avoir accès au maximum d'informations sur une entreprise. Par exemple les numéros de téléphone, télécopie, l'adresse, l'email, le site, etc. et ce grâce à un moteur de recherche « intelligent ».

Il était difficile de modéliser ces scripts en objet. L'objet étant un langage permettant une généralisation maximale, hors les scripts nécessaires devaient être les plus spécialisés possibles avec une liberté de modification tout de même. J'ai du me pencher sur la gestion avancée de base de données pour en assimiler certains principes.

2.32 Analyse des contraintes techniques

La première contrainte technique était de réaliser une BDD suffisamment souple pour pouvoir évoluer correctement. Il m'a fallu la segmenter au maximum pour avoir le plus de table et donc un maximum de combinaisons possibles. Mais la segmentation à un prix, elle complique la programmation (les requêtes sont assez imposantes).

Il m'a ensuite fallu trouver un format d'importation. Les données étaient déjà prêtes dans un fichier Excel, j'ai donc découvert le format CSV et j'ai du réaliser un importateur. Il m'a fallu optimiser au maximum le script, le fichier à importer étant relativement important (~ 3000 lignes).

J'ai ensuite été confronté à un problème de taille. En effet les requêtes a exécuter étant assez grandes, le temps de réponse du serveur commençait à devenir dangereux (~ 1.5 secondes dans le pire des cas, et ce pour une seule connexion, le site devant en supporter un maximum). J'ai donc du optimiser au maximum la BDD en utilisant des index.

J'ai du étudier les mécanismes de recherche de texte du serveur MySQL, et ainsi adapter la BDD pour permettre une recherche « intelligente ». Une fois de plus ce genre d'adaptation consommait trop de ressources, j'ai donc du optimiser une fois de plus.

Une dernière contrainte m'est apparue à la fin de la mission. Dans un premier temps j'ai voulu créer des tables relationnelles, pour éviter des données orphelines et avoir une BDD le plus propre et légère possible. Mais MySQL ne supporte pas le mécanisme de recherche sur les tables relationnelles. J'ai donc du faire un compromis entre une base qui risquait d'être « polluée » de données qui ne devraient plus exister, et entre une recherche intelligente. J'ai choisi de garder la recherche intelligente, il m'a semblé plus important de satisfaire l'utilisateur plutôt que d'avoir une base la plus propre possible (les données orphelines n'entravent que très peu la rapidité d'exécution).

2.33 Organisation

Cette mission c'est donc déroulée en 3 parties, j'ai tout d'abord analysé le contenu du portail, pour en évaluer les besoins en termes de BDD. Il m'a ensuite fallu créer un importateur pour la remplir correctement (en suivant le schéma des tables que j'avais préalablement défini). Une fois ces deux phases accomplis j'ai pu réaliser un moteur de recherche calqué sur le fonctionnement de vrai moteur de recherche web (comme Google par exemple).

J'ai donc commencé par analyser les données à enregistrer, puis j'ai réalisé une ébauche de BDD sur papier. J'ai ensuite modélisé la base de données grâce à un programme, ce qui m'a permis d'obtenir les requêtes de création de tables. J'ai du réécrire ces requêtes car le programme que j'ai utilisé était en version bêta et exportait très mal le rendu graphique (j'ai essayé un autre programme pour pallier à ce problème, mais le résultat n'a pas été franchement meilleur). J'ai ensuite créé les tables sur un serveur, mais je n'y ai pas ajouté d'optimisation.

J'ai ensuite réalisé l'importateur CSV en analysant la structure d'un fichier exemple. Une fois que l'importateur fonctionnait correctement avec ce fichier j'ai pu importer les vraies données. Il m'a bien sûr fallu faire des corrections lors de cette importation.

J'ai ensuite pu optimiser la BDD en me penchant sur le fonctionnement des index et en testant. Cela m'a également permis de maitriser un peu plus le langage SQL permettant d'effectuer les requêtes.

J'ai terminé par le moteur de recherche. Avant de commencer le code, il m'a d'abord fallu voir le fonctionnement des recherches avancées sur le serveur. Après avoir assimilé ces principes j'ai apporté la dernière couche d'optimisation à la BDD et j'ai commencé le code (qui n'est pas très compliqué se limitant à une succession de requêtes, j'ai choisi de ne pas en faire une sous partie).

2.34 La base de données

La base de données finale est composée des 9 tables suivantes :

- groupe_activite : Permet de créer des groupes d'activités pour classer les secteurs d'activités.
- secteur_activite : Répertorie les secteurs d'activités (taxis, hôtels, etc.). Peut-être classé dans un groupe d'activité, mais n'est pas obligatoire.
- entreprise : Les informations des entreprises. Comprend le nom, une description (facultative) et une adresse de site internet (facultative). Cette table comprend les informations uniques d'une entreprise.
- numero : Contient les numéros de téléphones, fax et portable. Un champ facultatif permet également de renseigner le poste (secrétariat, service infographie, etc.), un champ renseigne sur le type du numéro (fax, téléphone ou portable, ce champ est modifiable si besoin est), un dernier champ correspond à l'identifiant de l'entreprise à qui il appartient. Cette structure était nécessaire, une entreprise pouvant avoir plusieurs numéros.
- mail : Contient toute les adresses mail. Un champ renseigne le propriétaire de l'adresse (l'entreprise), un autre facultatif permet de connaître le poste (secrétariat, service infographie, etc.). Une entreprise pouvant avoir plusieurs adresses mail.
- canton : Les communes du pays bigouden sont classées par canton, cette table sert à en créer un.
- commune : Répertorie les communes. La table contient un champ pour le nom de la commune, un autre pour son code postal et un dernier pour son canton (facultatif).
- adresse : Contient toute les adresses. Cette table contient uniquement l'intitulé exact de l'adresse, ainsi qu'un champ qui lui associe sa commune.
- adresses_entreprises : Une entreprise peut avoir plusieurs adresses (par exemple, il peut y avoir un garage « speedy » à Brest et un autre à Quimper, ou 3 à Quimper…). Cette table est issue d'une relation entre la table adresse et la table entreprise. Elle ne contient que des clés étrangères.

Chaque table dispose également d'index. Un index permet au moteur de la base de données de ne pas avoir à parcourir les données pour faire une recherche ou une jointure. L'étude des

index m'a permis de grandement optimiser mes requêtes. Par exemple si il y a une jointure entre deux tables chargées (15000 enregistrements pour la première, 20000 pour la seconde), sans index le moteur aura 15000 * 20000 enregistrements à parcourir. Alors qu'avec des index sur les clés utilisées pour la jointure le nombre d'enregistrements à parcourir sera de 1 (en réalité le moteur ne parcours même pas les données, il parcourt juste le fichier d'index pour localiser l'emplacement de la donnée à extraire). Avec une base de données relationnelle il est obligatoire d'indexer d'abord les champs avant de créer les clés étrangères. Il faut également faire attention à ne pas indexer trop de champs, le serveur serait trop vite saturé et perdrait des performances.

Certains champs possèdent également un type d'index spécifique, il s'agit des index « fulltext ». Ce sont des index spécifiques pour les champs textes, permettant d'effectuer une recherche (voir sous partie 2.36).

2.35 L'importateur CSV

Tout d'abord j'ai du étudier la structure d'un fichier CSV. Il s'agit d'un fichier texte représentant les cellules d'une table Excel. Le fichier commence par la déclaration des champs (on peut donc compter le nombre de champs, et garder uniquement ceux que l'on veut), ensuite chaque ligne correspond à une ligne de la table Excel, les cellules sont (habituellement) séparées par des points-virgules (;).

Voici la structure du script que j'ai établi :

- Définition des champs à importer, il s'agit d'énumérer les champs voulus.
- Ouverture du fichier.
- Recherche des champs, calcul des index (position de la donnée dans la ligne, son numéro de cellule).
- Parcours de chaque ligne du fichier, on ne garde que les données qui se situent aux index trouvés. On ignore également les champs vides ou les champs remplient de caractères interdits (définis par l'utilisateur au début du fichier). Toutes les informations sont stockées dans un tableau en PHP (en mémoire).
- Fermeture du fichier.
- Enregistrement des données dans la BDD. Cette phase se découpe comme cela :

- Secteur d'activité.
- Entreprise.
- Adresse mail.
- Numéro de portable, de téléphone, ou de fax.
- Commune et code postal.
- Adresse.
- Liaison entreprise – adresse.

L'enregistrement est obligé de suivre cet ordre, des données étant nécessaire dans d'une table avant l'autre (la table entreprise doit connaître le secteur d'activité avant de pouvoir enregistrer par exemple).

Ces phases sont décomposées comme ceci :

- Vérification que la donnée n'existe pas dans la BDD. Si c'est le cas on récupère sont identifiant.
- Sinon on insère la donnée et on récupère le nouvel identifiant.

A la fin de l'importation l'utilisateur est informé des erreurs qui se sont produites (tentative d'insertion d'un texte à la place d'un numéro de téléphone par exemple), et du nombre d'enregistrements effectués.

2.36 Le moteur de recherche

Le plus compliqué restait le moteur de recherche. Durant mes recherches j'ai pu constater que les méthodes utilisés traditionnellement pour rechercher du texte dans une base de données n'étaient pas optimisés (n'utilisant pas les index, chaque enregistrement est parcouru). J'ai donc découvert le type d'index « fulltext », il s'agit d'un index qui répertorie les mots les plus courants. En effectuant une recherche fulltext le moteur de la BDD ouvre l'index et cherche les mots demandés, il tri ensuite les résultats par ordre de pertinence. Cela permet d'avoir une sorte de moteur de recherche « intelligent ». Le moteur de recherche Google fonctionne un peu sur le même principe, chaque page à un score (le page rank), et un index garde les mots, ainsi que les liens des pages. Lors d'une recherche les index sont parcourus, et les sites qui ont un fort page rank et dont les mots ont été trouvé dans l'index sont affichés en premier.

De plus ce genre de recherche possède un seuil. Il s'agit du seuil des 50%, par exemple si lors d'une recherche un mot est trouvé très fréquemment (comme « de », « le », etc.) il aura un fort taux de pertinence. Et bien le moteur supprime les résultats dès qu'ils dépassent 50% du résultat total, pour éviter d'avoir des résultats parasites.

Il est également possible de consulter le résultat de pertinence d'une recherche, c'est cette technique que j'ai utilisé pour le moteur de recherche. J'attribue un score pour chaque table (la requête utilisée pour la recherche « fusionne » presque toute les tables, pour obtenir les différentes possibilités). Par exemple la table adresse à un score et la table entreprise aussi, lorsqu'un utilisateur recherche un nom d'entreprise, le score de la table adresse sera faible (0 si le mot n'est pas contenu dans l'adresse), alors que celui de l'entreprise aura un score élevé. En additionnant ces scores ont obtient le résultat le plus pertinent possible. Toutefois il peut y avoir des résultats parasites (si l'utilisateur recherche « quimper », le moteur lui donne comme réponse les adresses qui sont composées de quimper, comme « 1 route de quimper »).

```
SELECT
@score_entreprise := MATCH (entreprise, description, site) AGAINST
(@recherche) AS score_entreprise,
entreprise,
description,
site,
@score_secteur_activite := MATCH (secteur) AGAINST (@recherche) AS
score_secteur_activite,
secteur,
@score_adresse := MATCH (adresse) AGAINST (@recherche) AS score_adresse,
adresse,
@score_commune := MATCH (commune) AGAINST (@recherche) AS score_commune,
commune,
code_postal,
@score_entreprise + @score_secteur_activite + @score_adresse +
@score_commune AS score
FROM entreprise
LEFT JOIN secteur_activite ON secteur_activite.id = id_secteur_activite
LEFT JOIN adresses_entreprises ON id_entreprise = entreprise.id
LEFT JOIN adresse ON adresse.id = id_adresse
LEFT JOIN commune ON commune.id = id_commune";
GROUP BY adresse
HAVING score NOT LIKE 0
ORDER BY score DESC
```

```
LIMIT 30

Voici la requête principale qui effectue la recherche. On récupère les
scores, et les informations de l'entreprise. On calcule le score total pour
chaque ligne, on enlève les scores nuls puis on tri les résultats par
score, on termine en limitant le nombre de résultats à 30 (au-delà la
recherche n'est pas bonne).
```

En plus de cette recherche j'ai ajouté une sous-recherche des codes postaux. Lorsque le script détecte qu'un code postal a été tapé (5 chiffres à la suite), il affine la recherche en ne sélectionnant que les entreprises situées dans cette/ces commune(s). De plus il affiche un lien vers les communes qui ont ce code postal.

Puis j'ai rajouté un annuaire inversé. Si le script détecte 5 séries de deux chiffres séparés par un espace ou un point il cherche le numéro correspondant dans la base de données, s'il trouve un résultat (logiquement il n'y a qu'un seul résultat), il affiche les informations de l'entreprise. Lors de la recherche d'un numéro, le moteur de recherche n'utilise pas la recherche « intelligente », il m'a semblé illogique de rechercher des adresses, entreprises, etc. avec un numéro de téléphone (cela aurait été déroutant pour l'utilisateur en plus).

Lors de l'affichage des résultats, les mots trouvés sont également affichés en gras, pour permettre une meilleure identification.

2.37 Problèmes persistants

Comme dit précédemment, MySQL ne supporte pas les index fulltext sur une table relationnelle, il peut donc y avoir une surcharge de la base de données (mais c'est très négligeable). Toutefois j'ai conçu les requêtes de création de table de manière à y inclure les relations (le serveur les ignores) en attendant qu'une version de MySQL gérant les index fulltext sur les tables relationnelles sorte (ce qui est peu probable, les index consomment beaucoup de ressources lors d'insertion de données, et la gestion des relations n'est pas évidente non plus).

L'importeur fonctionne, mais à ses limites. Il ne peut pas comprendre que « speedy brest » et « speedy » sont la même entreprise, il peut donc y avoir une légère surcharge de la base de données (une alternative aurait pu être envisagée, mais les traitements étaient trop conséquents).

La recherche en fulltext est très efficace… mais principalement sur de longs textes, sur de tout petit texte (comme ici) les résultats sont souvent parasités. Toutefois d'après mes tests les 3 premiers résultats étaient assez pertinents.

J'ai rencontré un problème lors de la conception de la requête. Le moteur de la BDD tenait compte des majuscules (alors que la documentation spécifiait l'inverse). Il m'a donc fallu envisager les possibilités : une recherche en minuscule, une recherche avec la première lettre en majuscule, une recherche en majuscule. Mais il était hors de question pour moi d'effectuer 3 requêtes si conséquentes d'affilé. La solution était d'utiliser un OU logique dans le champ de recherche. Les résultats sont nettement plus satisfaisant avec cette méthode, mais il peut y avoir encore plusieurs résultats parasites, ou pire : des résultats ignorés (sans cette technique la recherche de « blue ouest » ne donnait rien, alors que « Blue Ouest » si).

Le même problème se pose avec les accents, mais je n'ai pas souhaité le régler (trop de traitement en PHP, et surtout trop de termes à rechercher ensuite, ce qui aurait ralenti la requête).

CONCLUSION

Ces trois missions m'ont été grandement bénéfiques. La première m'a permis de réaliser des scripts en programmation traditionnelle, et de constater que la programmation orientée objet aurait été bien plus efficace. De plus j'ai enfin pu apprendre à me servir des expressions régulières.

La deuxième mission m'a permis de réellement essayer la programmation orientée objet et surtout l'Ajax qui devient de plus en plus répandu dans les nouvelles applications. Cette pseudo-évolution est très pratique pour développer des applications légère (l'inverse d'un client lourd). Ce genre d'application va énormément se développer dans le futur, SUN Microsystem avait développé JAVA dans cette optique, mais la lenteur du réseau avait freiner l'évolution. La programmation orientée objet permet de concevoir des applications rapidement, et ajoute une certaine souplesse de programmation.

La troisième mission m'a permis d'approfondir ma connaissance en SQL et en SGBD. J'ai pu apprendre à optimiser les tables et les jointures. L'étude d'un moteur de recherche m'a permis d'apprendre beaucoup sur le fonctionnement des recherches sur les chaines de caractères (comme l'algorithme de Levenshtein).

Ce stage m'a parfaitement convenu, on m'a confié des missions intéressantes, et qui m'ont permis d'avancer à mon rythme (il n'y avait pas réellement de contrainte de temps).

REFERENCES BIBLIOGRAPHIQUES

PIERRE DE GEYER Cyril & DASPET Eric, *PHP 5 avancé*, Editions Eyrolles, Juin 2004

Ajaxload : www.ajaxload.info, permet de générer une image pour un indicateur d'activité AJAX.

Google : http://www.google.fr, le célèbre moteur de recherche. Une référence.

Manuel officiel de PHP : http://fr.php.net/manual/fr/

Manuel officiel de MySQL : http://dev.mysql.com/doc/refman/5.0/fr/

NeoBlog : http://blog.neovov.com, mon blog personnel, vous y trouverez des explications des scripts que je développe et des réflexions sur l'ergonomie web.

Nexen.net, Portail PHP et MySQL : www.nexen.net, propose des articles sur PHP et MySQL et des traductions de documentation.

Open web : http://openweb.eu.org, propose des articles sur les standards, ainsi que tout ce qui concerne les évolutions récentes du web.

Phpinfo.net : http://www.phpinfo.net, propose des articles avancés sur PHP et MySQL.

Self HTML : http://fr.selfhtml.org, propose une documentation simple sur l'utilisation de JavaScript pour le DOM.

Wikipédia : http://fr.wikipedia.org, encyclopédie communautaire.

GLOSSAIRE

AJAX : *Asynchronous Javascript And XML* (XML et Javascript asynchrones). Acronyme désignant une méthode informatique de développement d'applications Web.
AJAX n'est pas une technologie en elle-même, mais un terme qui évoque l'utilisation conjointe d'un ensemble de technologies couramment utilisées sur le Web.

ASCII : *American Standard Code for Information Interchange*. Cette norme a longtemps été utilisée pour le codage des caractères alphanumériques en informatique. Encore aujourd'hui, la table ASCII est grandement utilisée, même si parfois complétée par une table étendue.

Base de données relationnelle : Type de base de données permettant de lier des tables entre elles. Cela permet une meilleure gestion des données et évite de se retrouver avec des données orphelines, dénuées de sens dans le flot de données.

Blog : Site web sur lequel une ou plusieurs personnes s'expriment de façon libre, sur la base d'une certaine périodicité. Son expression est décomposée en unités chronologiques ; chaque unité est susceptible d'être commentée par les lecteurs et est le plus souvent enrichie de liens externes. Le mot blog est né de la contraction de «*web log*» (c'est-à-dire *carnet de bord Web*.) Contrairement au site personnel, le blog bénéficie d'une structure éditoriale préexistante, sous la forme d'outils de publication plus ou moins formatés.

C++ : Langage de programmation permettant la programmation sous de multiples paradigmes comme, par exemple, la programmation procédurale, la programmation orientée objet et la programmation générique. Le langage C++ n'appartient à personne et par conséquent n'importe qui peut l'utiliser sans payer de droits.

Classe : En programmation orientée objet, une classe déclare des propriétés communes à un ensemble d'objets. La classe déclare des attributs représentant l'état des objets et des méthodes représentant leur comportement.

Clé étrangère : Indice permettant au SGBD de faire la relation entre deux tables.

CMS : *Content Management System* (systèmes de gestion de contenu). Famille de logiciels de conception et de mise à jour dynamique de sites web.

CSS ou Feuilles de style : *Cascading Style Sheets*. Utilisé pour décrire la présentation d'un document structuré écrit en HTML ou en XML, et c'est le W3C qui en a la direction.

Débogage : le débogage est l'éradication des anomalies dans un programme informatique l'empêchant de fonctionner correctement. Leur gravité peut aller de bénigne (défauts d'affichage mineur) à majeure (explosion du vol 501 de la fusée Ariane 5).

DOM : *Document Object Model*. Recommandation du W3C qui décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents. Le document peut ensuite être traité et les résultats de ces traitements peuvent être réincorporés dans le document tel qu'il sera présenté.

Expression régulière ou Expression rationnelle : Famille de notations compactes et puissantes pour décrire certains ensembles de chaînes de caractères. Permet de parcourir des textes à la recherche de morceaux de texte ayant certaines formes, et éventuellement remplacer ces morceaux de texte par d'autres.

Firefox : Navigateur Web libre développé et distribué par la Fondation Mozilla aidée de centaines de bénévoles grâce aux méthodes de développement open source. Avant de se concentrer sur Firefox, Mozilla était surtout connu pour sa suite intégrée de logiciels Internet, d'une conception plus ancienne héritée de Netscape.

Flux RSS : *Really Simple Syndication* (syndication vraiment simple). Format de syndication de contenu Web.

Héritage : Principe de la programmation orientée objet, permettant entre autre la réutilisabilité et l'adaptabilité des objets. Elle se nomme ainsi car le principe est en quelques sorte le même que celui d'un arbre généalogique, on les appelle *classes mères*. Ces classes possèdent des attributs et/ou des méthodes qui leurs sont propres, et qui pourront être transmis aux classes filles découlant des classes mères.

HTML : *Hypertext Markup Language*. Langage informatique créé et utilisé pour écrire les pages Web. HTML permet en particulier d'insérer des liens dans du texte, donc de créer de l'hypertexte, d'où le nom du langage.

Hypertexte : Un système hypertexte est un système contenant des documents liés entre eux par des liens permettant de passer automatiquement (en pratique grâce à l'informatique) du document consulté à un autre document lié. Un document hypertexte est donc un document qui contient des liens.

Internet Explorer : Internet Explorer, parfois abrégé IE, MSIE ou MS IE, est le navigateur Web propriétaire de Microsoft, vendu et installé par défaut avec Windows.
C'est, depuis la fin des années 1990, le navigateur le plus utilisé au monde, bien que sa suprématie soit de plus en plus contestée par des navigateurs tels que Firefox.

JAVA : Technologie composée d'un langage de programmation orienté objet et d'un environnement d'exécution.

JavaScript : Langage de programmation de type script, utilisant les objets, principalement utilisé dans les pages Web.

Lecteur d'écran : Logiciel qui tente d'identifier et d'interpréter ce qui est affiché sur un écran, ensuite le texte est retranscrit, soit en un texte braille, soit en un texte sonore (par synthèse vocale). C'est un logiciel utilisé pour les déficients visuels.

Méthode : En programmation orientée objet, la méthode est une fonction faisant partie de l'interface d'un objet. Dans ce cas, les méthodes sont aussi appelées des *méthodes d'instance* et n'agissent que sur un seul objet à la fois. D'autres méthodes, appelées *méthodes de classe* ou *méthodes statiques*, permettent d'agir sur tous les objets d'une même classe.

Méthode GET : Méthode de transmission de variables et de leurs données en utilisant l'URL.

Merise : Famille de méthodes du domaine informatique. Elle est constituée essentiellement d'une méthode d'analyse et de conception et d'une méthode de gestion de projet.

MySQL : Gestionnaire de base de données libre et gratuit. Il est très utilisé et très populaire dans les projets libres. Il est néanmoins très utilisé dans le milieu industriel également grâce à son rapport qualité/prix.

Objet : On appelle objet l'instance d'une certaine classe.

Opera : Navigateur Web alternatif. Ses fonctionnalités et sa puissance en font un navigateur pratique pour les développeurs de sites web. Il supporte de grosses charges (40 pages ouvertes peuvent se charger en même temps), et fait partie des navigateurs attachés à respecter les règles du W3C.

PHP : *Hypertext Preprocessor* (acronyme récursif). Langage de script libre principalement utilisé pour être exécuté par un serveur HTTP, mais il peut fonctionner comme n'importe quel langage interprété en utilisant les scripts et son interpréteur sur un ordinateur. PHP permet de développer des scripts suivant le modèle procédural et/ou le modèle objet. En raison de la richesse de sa bibliothèque, on désigne parfois PHP comme une plate-forme plus qu'un simple langage.

Portée des variables ou Portée des méthodes : En programmation orienté objet, la portée d'une variable est l'étendu des accès qu'elle autorise. Une variable *privée* ne sera accessible que dans la classe, alors qu'une variable *public* sera accessible de n' importe où. Les mêmes règles s'appliquent à la portée des méthodes.

Programmation Orientée Objet : Façon d'architecturer une application informatique en regroupant les données et les traitements sur ces dernières au sein d'une même entité, les objets.

Prototype : Le prototype d'une fonction (d'une procédure ou d'une méthode) désigne la syntaxe d'appel de celle-ci.

Safari : Navigateur Web pour Mac OS X développé par Apple.

Sémantique : Comme en linguistique, ici la sémantique des langages informatiques désigne le lien entre un signifiant, le programme, et un signifié, objet mathématique qui dépendra des propriétés que l'on souhaite connaître du programme.
On appellera aussi *sémantique* le lien entre le langage signifiant (le langage de programmation) et le langage signifié.

Serveur HTTP ou **serveur Web :** Logiciel servant des requêtes respectant le protocole de communication client-serveur HyperText Transfer Protocol (HTTP), qui a été développé pour le World Wide Web.

SGBD : Système de gestion de base de données. La gestion et l'accès à une base de données sont assurés par un ensemble de programmes qui constituent le Système de gestion de base de données.
Un système de gestion de bases de données héberge généralement plusieurs bases de données, qui sont destinées à des logiciels ou des thématiques différentes.

Standards du Web : Les standards du Web sont un ensemble de technologies et de protocoles utilisés sur le Web définis par le W3C sous forme de recommandations. La tâche de proposition, définition et d'acceptation de nouveaux standards du Web (ou d'amélioration de standards déjà existants) est confiée au W3C.

Système de fichiers : En informatique, un système de fichiers est une méthode d'organisation des données persistantes sur un médium durable (par exemple : disque dur, disquette, CDROM, clef USB ...).
Le système de fichiers offre à l'utilisateur une vue abstraite sur ses données. L'unité de stockage est le *fichier*, qui est une séquence d'octets ; les fichiers sont groupés dans des collections nommées *répertoires* ; les répertoires sont organisés en arborescence (il y a un répertoire racine et des sous-répertoires).

Template (anglicisme) : Terme utilisé en informatique pour désigner un modèle de conception de logiciel ou de présentation des données. On parle aussi de « patron » comme en couture ou de gabarit.

UML : *Unified Modelling Language* (langage de modélisation unifié). Langage graphique de modélisation des données et des traitements. C'est une formalisation très aboutie et non-propriétaire de la modélisation objet utilisée en génie logiciel.

UNIX : UNIX est le nom d'un système d'exploitation créé en 1969, à usage principalement professionnel, conceptuellement ouvert et fondé sur une approche par laquelle il offre de nombreux petits outils chacun dotés d'une mission spécifique, multitâche et multiutilisateur. Il a donné naissance à une famille de systèmes, dont les plus populaires en 2006 sont GNU/Linux, *BSD et Mac OS X.

Upload : En informatique, le téléchargement est l'opération de transmission d'informations — programmes, données, images, sons, vidéos — d'un ordinateur à un autre via un canal de transmission. L'upload est l'action d'envoi de données par l'utilisateur sur une machine distante.

URL : *Uniform Resource Locator* (repère uniforme de ressource). Chaîne de caractères utilisée pour identifier les ressources dans le World Wide Web : document HTML, image, son, boîte aux lettres électronique... Elle est informellement appelée une adresse Web ou URL.

W3C : *World Wide Web Consortium*. Consortium fondé en octobre 1994 pour promouvoir la compatibilité des technologies du World Wide Web. Le W3C n'émet pas des normes, mais des recommandations.

Wiki : Site Web dynamique permettant à tout individu d'en modifier les pages à volonté. Il permet non seulement de communiquer et diffuser des informations rapidement, mais de *structurer* cette information pour permettre d'y naviguer commodément. Un éditeur Wiki est un éditeur de texte permettant à quiconque de modifier du contenu sans connaître la programmation Web.

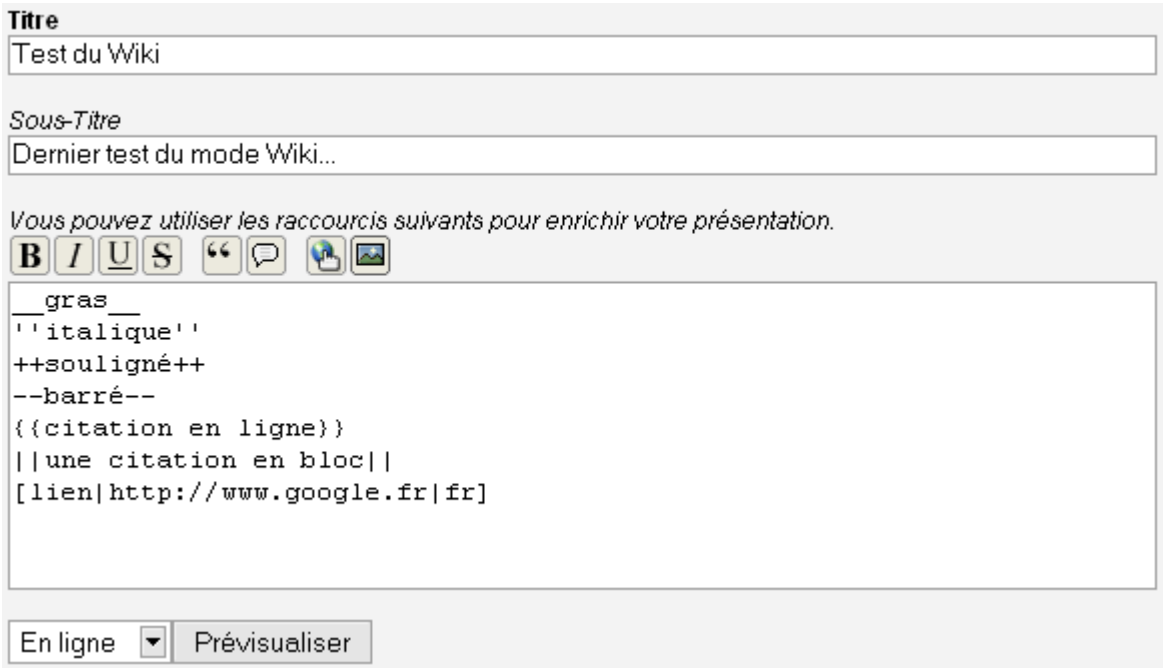
Windows : Windows est une gamme de systèmes d'exploitation produite par Microsoft, principalement destinées aux compatibles PC. Depuis les années 1990, avec la sortie de Windows 95, son succès commercial pour équiper les ordinateurs personnels est tel qu'il possède un statut de quasi-monopole.

xHTML : Langage balisé servant à l'écriture de pages du World Wide Web. xHTML est le successeur de HTML et respectant la syntaxe définie par XML, plus récente et plus simple que la syntaxe définie par SGML respectée par HTML.

XML : *Extensible Markup Language* (langage de balisage extensible). Standard du W3C qui sert de base pour créer des langages de balisage : c'est un « méta-langage ». En ce sens, XML permet de définir un vocabulaire et une grammaire associée sur base de règles formalisées. Il est suffisamment général pour que les langages basés sur XML, appelés aussi dialectes XML, puissent être utilisés pour décrire toutes sortes de données et de textes.

ANNEXES

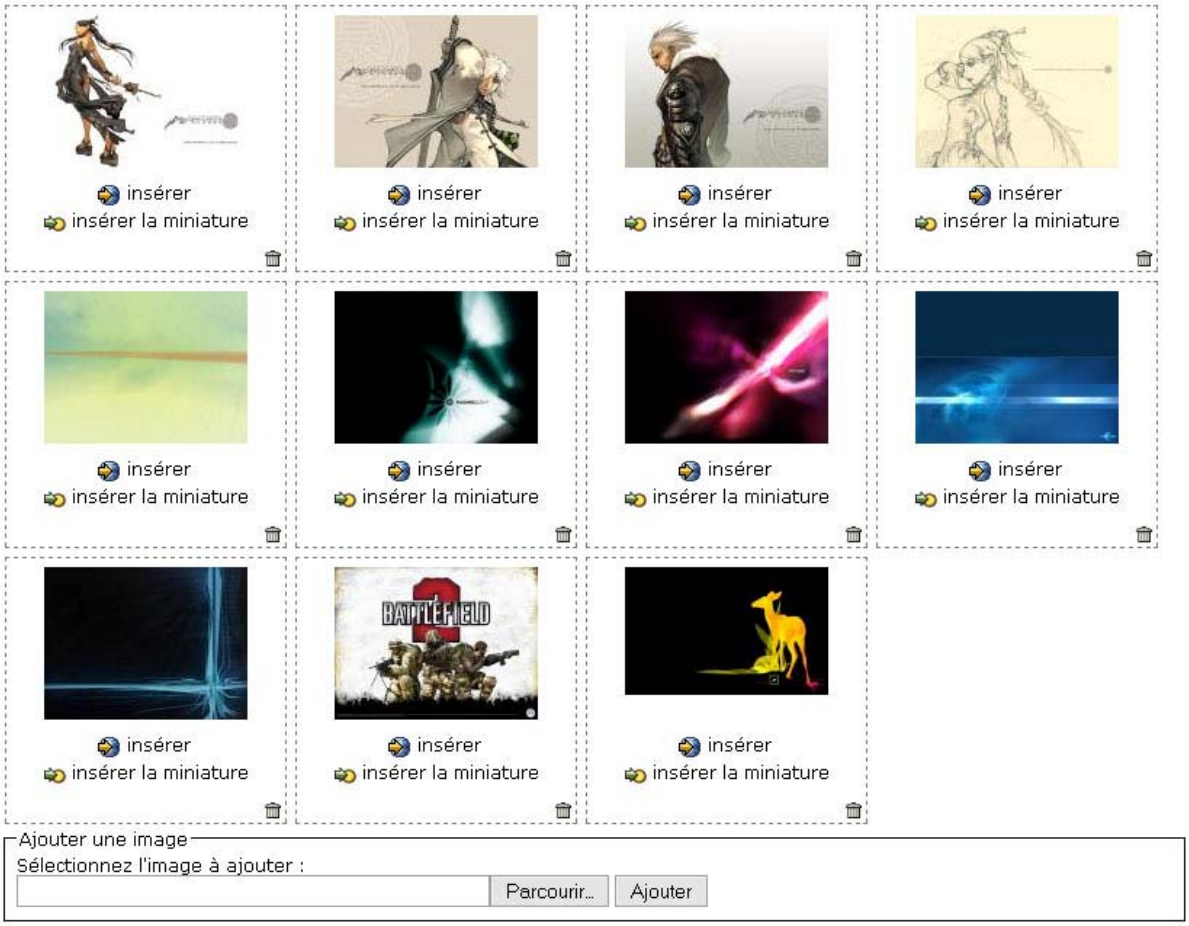
I – Le gestionnaire de nouvelles



L'éditeur Wiki. Du texte à déjà été inséré pour montrer la syntaxe

Le rendu sur un site de test (réalisé rapidement pour tester la facilité de mise en forme et la rapidité d'utilisation des templates)

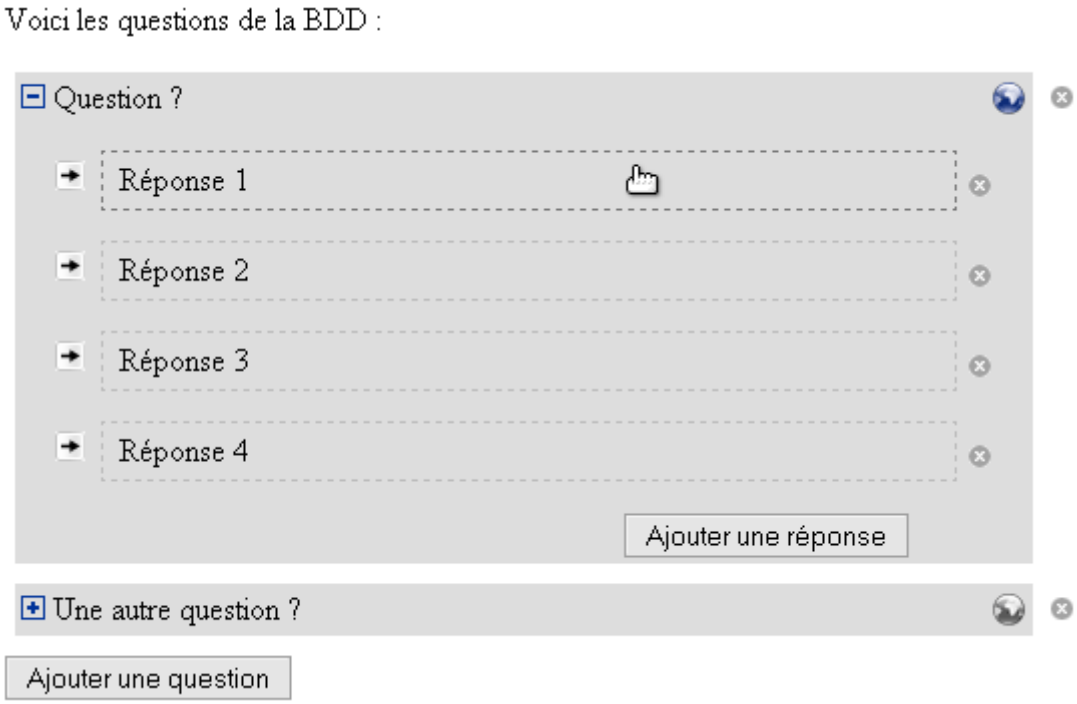
Le module intégré dans cinéland (projet de fin d'année de l'année dernière). L'intégration a pris moins de 5 minutes.



Copie d'écran du gestionnaire d'image.

II – Le gestionnaire de sondage

Ma classe MySQL modélisée en UML.
Vous pouvez consulter la documentation complète à cette adresse :
http://www.neovov.com/res/classes/MySQL/doc/
ou la télécharger à cette adresse :
http://www.neovov.com/res/classes/MySQL/doc/MySQL.pdf



Le rendu final du gestionnaire de sondage.
Un simple click sur un champ permet de le changer et de l'enregistrer.

L'indicateur d'activité.
Informe l'utilisateur que le script récupère ou sauvegarde des informations en arrière plan.

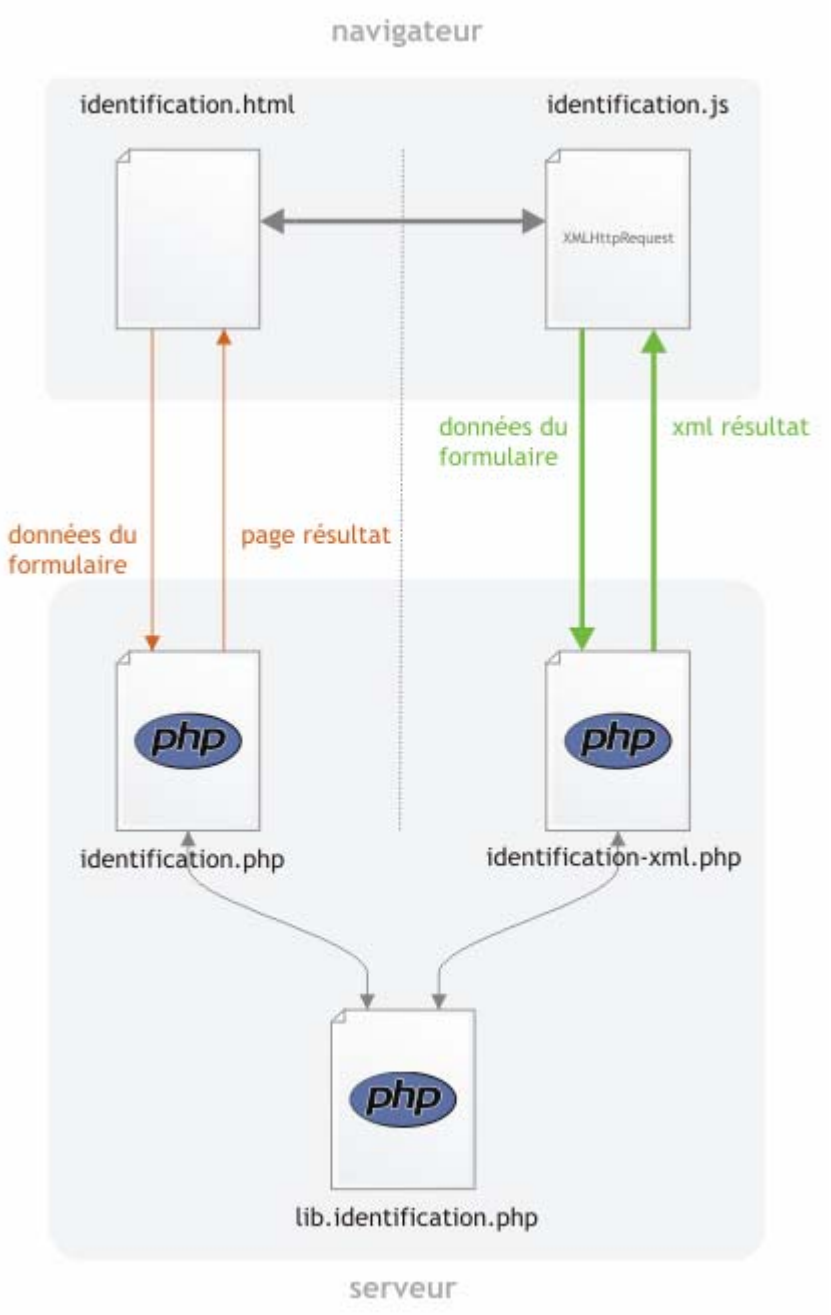


Schéma des transactions d'une page utilisant de l'AJAX.
A gauche la structure habituelle, à droite le fonctionnement grâce à JavaScript et l'objet XMLHttpRequest.
Pour des exemples concrets et une très bonne explication je conseille de lire l'article de Maurice Svay sur Open Web :
http://openweb.eu.org/articles/objet_xmlhttprequest/

Base de données finale. Les index, clés fulltext et clés étrangères y sont indiqués.
Les relations entre les tables ne sont pas affichées, le programme ne fonctionnant pas correctement.

Fiche Résumé

Résumé

Durant ce stage j'ai pu appréhender la programmation orientée objet et l'Ajax. J'ai également pu approfondir mes connaissances dans les bases de données et le PHP. Durant les trois missions qui m'ont été confiés j'ai pu réaliser un module de gestion et de publication de nouvelles, un gestionnaire de sondage, une base de données conséquente et un moteur de recherche.

Mots-clés

Edition, site web, gestionnaire de nouvelles, gestionnaire de sondage, moteur de recherche, programmation orientée objet, Ajax, DOM, Wiki, base de données.