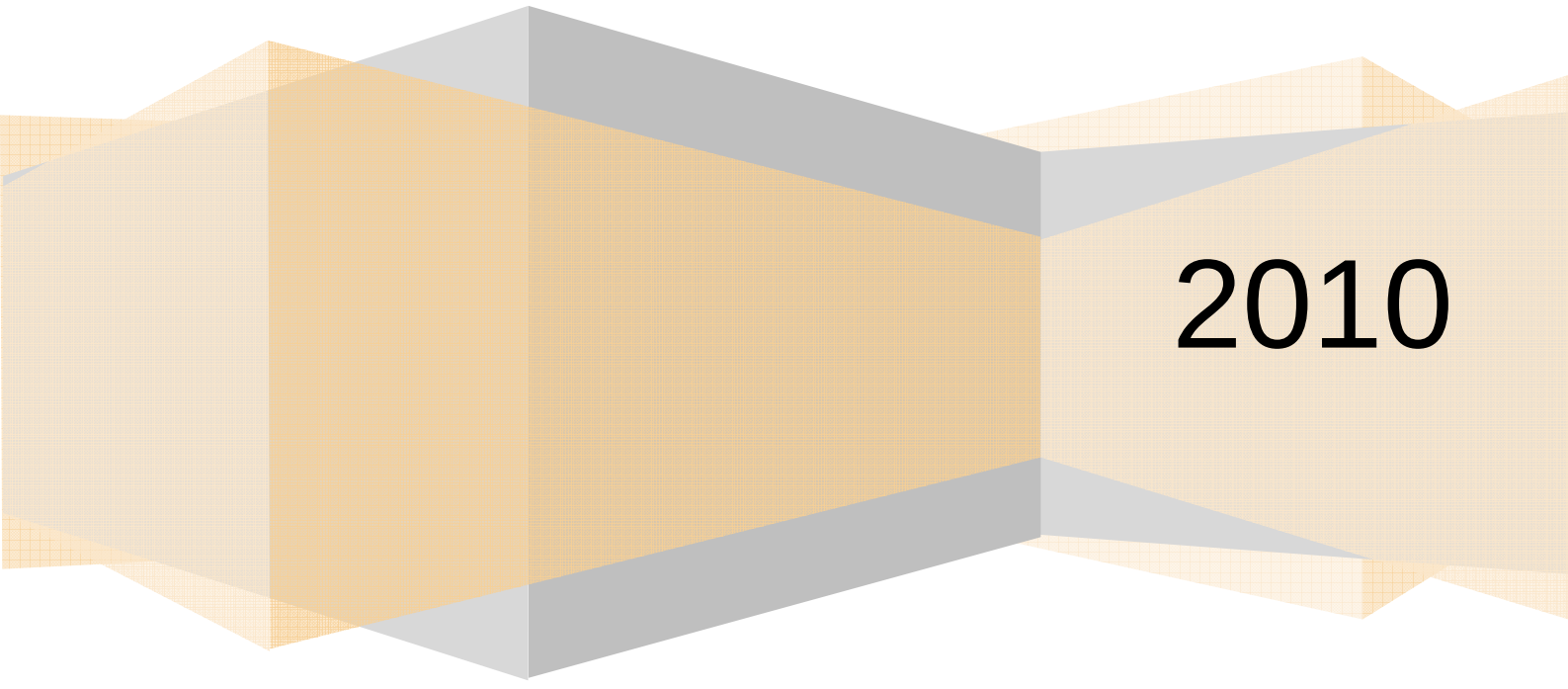
Société D2E

Migration d’une application Access vers une application Web riche

Mémoire de fin d’étude

Par Antoine Sassoust



2010

Table des matières

Introduction

Ce mémoire va décrire un projet que j'ai du mener à terme lors de mon stage de fin d'études à Supinfo. C'est donc naturellement que je vais vous présenter l'entreprise dans laquelle j'ai évolué, en parlant un peu de son fonctionnement interne. Puis je présenterai le logiciel qu'elle développe ainsi que les marchés qu'elle cible.

Ensuite, je vous expliquerai en quoi à consisté mon travail, c'est-à-dire la problématique qui m'a été présentée et les différentes étapes de mon projet.

Je vais expliquer comment j'ai procédé pour faire fonctionner l'ancien logiciel à migrer sur une version Web, en essayant de le modifier le moins possible.

Puis nous étudierons les différents outils qui se présentaient à nous pour développer une nouvelle application Web riche, et pourquoi certains ont été retenus plutôt que les autres dans le cadre de ce projet.

Enfin, j'expliquerai la réflexion que j'ai eu en amont du projet pour réaliser un outil permettant de générer rapidement des listes de données sous forme d'écrans web, afin de migrer l'ancienne application rapidement et avec des outils la rendant encore plus intéressante.

Pour finir, je ferai le bilan de mon stage, et remercierai les personnes m'ayant permis de réaliser ce projet, avec qui j'ai travaillé ou qui m'ont soutenu.

1. D2E,

Une société tournée vers l’avenir, misant sur des outils technologiques modernes.

1.1 Présentation

La société D2E est une société de services créée en 2003 afin de développer et commercialiser un logiciel de gestion vitivinicole nommé process2wine. Deux personnes se trouvent à l'origine de cette entreprise, Mr. Emmanuel Thibierge et Mr. Alain Sutre. Depuis sa création, l'entreprise n'a cessée de se développer. L'équipe et les locaux se sont agrandis, ainsi que les secteurs ciblés. Nous allons ici présenter l'entreprise, depuis son origine jusqu'à ses projets à venir, en passant par les conditions de travail, l'équipe et les marchés sur lesquels elle évolue et ceux où elle voudrait évoluer.

1.2 Histoire de l'entreprise

En 1998, Mr Alain Sutre, qui est à cette époque à la tête du groupe de châteaux de vin « Le Taillan », décide de mettre en place un projet de gestion de qualité pour ses châteaux. Un programme sous Microsoft Access est alors développé, il se décompose en plusieurs modules distincts qui permettent de gérer l'ensemble d'un château.

Puis, en Juin 2000, Alain Sutre crée une entreprise nommée ERTUS, de consultants spécialisés dans le secteur agricole, viticole et vinicole. C'est alors que Mr. Emmanuel Thibierge devient consultant spécialisé dans les réseaux pour cette entreprise.

Après quoi, en 2003, Emmanuel Thibierge décide de créer sa propre entreprise : ET & Co, une SARL qui reste dans un cercle très familial. Cette entreprise de service a travaillé pour la chaine de Cafétérias

Crescendo afin de leur fournir une application intranet pour gérer leurs 53 restaurants en France.

Enfin, quelques mois plus tard, Emmanuel Thibierge et Alain Sutre décident de s'associer pour racheter les divers programmes sous Access au groupe Taillan, afin de les rassembler et les redévelopper d'une manière plus évolutive sous forme d'application web, et de commercialiser cette nouvelle version. Ils créent donc la SAS D2E avec un capital initial de 40,000€ afin de débuter le produit qui s'appellera process2wine.

1.3 Environnement

La société D2E est décomposée en 3 bureaux se trouvant en 2 endroits différents. Le siège social se situe à Bordeaux aux locaux de la société ERTUS que dirige toujours Alain Sutre, tandis que les bureaux pour l'équipe de développement et l'équipe de commerciaux se trouvent à Cenon avec Emmanuel Thibierge.

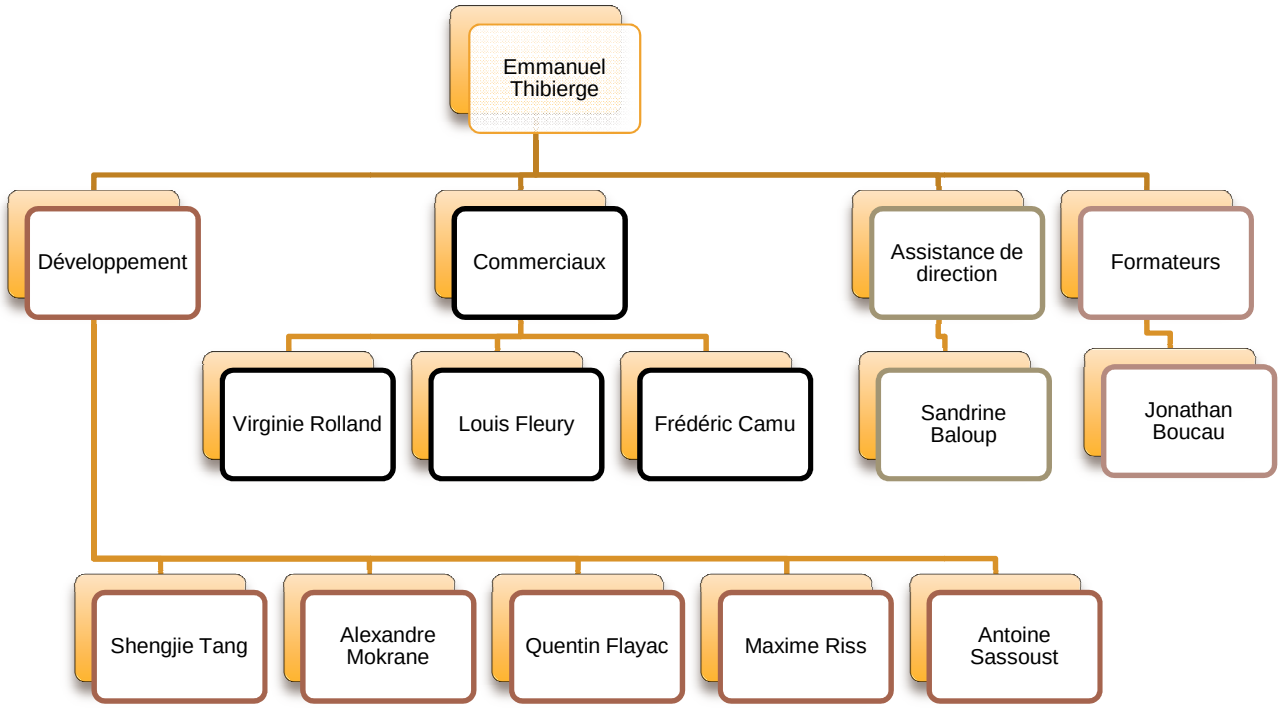
Les bureaux de développement font partis d'un complexe de bureaux à louer s'appelant Emeraude II. Cette structure a permis à la société de changer de bureau facilement pendant son évolution et l'agrandissement de son équipe. Dorénavant, D2E possède 2 bureaux dans ce complexe, le plus grand du bâtiment pour l'équipe de développement, et un petit bureau pour les commerciaux.

La société possède sa propre tour de serveurs, composée de 2 serveurs, 2 Switch et 2 Free box. Ces serveurs permettent de stocker les applications intranet et tous les outils utiles au développement (SVN, Wiki, Serveur de fichier, base de données, etc.).

Au niveau informatique, l'entreprise possède 5 PC fixes utilisant Windows 7 ou Ubuntu, ainsi que 5 ordinateurs portables sous Windows XP. Ces ordinateurs sont répartis selon les besoins de l'équipe (les commerciaux ont des ordinateurs portables, alors que l'équipe de développement et l'assistante de direction ont des ordinateurs fixes.).

1.4 Equipe

Voici ci-dessous, l'organigramme de la société D2E à Cenon.



1.4.1 Emmanuel Thibierge

Emmanuel Thibierge dirige la les lieux en tant que Directeur de production. C'est la personne qui prend toutes les décisions importantes et par laquelle il faut passer si on veut modifier un point important de la structure du programme.

Cependant, il a plusieurs rôles dans la société. Il est également formateur auprès des clients, car c'était à l'origine la seule personne compétente au niveau métier pour expliquer le fonctionnement du logiciel aux clients avant l'arrivée de Jonathan Boucau.

Coté développement, il fait office de chef de projet. C'est lui qui définit les priorités des travaux, et qui décrit les mécanismes métier qui doivent être compris afin de pouvoir développer certains écrans.

1.4.2 Les commerciaux

Les commerciaux agissent sur deux secteurs différents. Louis Fleury et Virginie Rolland se partagent le Bordelais, le Médoc et Saint-Emilion, tandis que Frédérique Camu s'occupe du Languedoc Roussillon.

Les commerciaux ne sont pas des formateurs, mais ils disposent d'une bonne connaissance métier, afin de prouver que le programme répond aux attentes des clients.

Ils possèdent l'accès à un site de démonstration mis à jour régulièrement avec la dernière version du programme, afin de faire visualiser les possibilités du programme aux clients et montrer les avantages que nous possédons face aux concurrents.

1.4.3 L'assistance de direction

Sandrine Baloup est l'assistante de direction d'Emmanuel Thibierge. Elle s'occupe du standard téléphonique, afin de rediriger les appels vers les bonnes personnes.

Elle s’occupe également de gérer tous les dossiers administratifs avec les sociétés extérieures (Cabinets d’expert comptables, Société d’hébergement, etc.).

C’est aussi elle, qui va noter les présences des employés afin d’envoyer les informations nécessaires aux comptable pour les bulletins de paie. Et qui est en charge du suivi des paiements des clients. Elle doit s’assurer que les paiements ont bien été envoyés par ceux-ci et se charge des courriers pour les banques et la poste.

1.4.4 Les formateurs

Jonathan Boucau est le premier formateur engagé pour aider Emmanuel Thibierge auprès des clients. C’est une personne ayant travaillé dans les châteaux auparavant, et qui a donc une bonne connaissance métier pour la création de vin.

Son rôle est très important car il va pouvoir, à la fois former les clients sur le programme, remonter les divers bugs d’utilisation poussée du logiciel, aider à résoudre les problèmes avec les clients en hot line et définir en partie les projets urgents à faire avancer.

C’est la personne qui fait office de chef de projet pour l’équipe de développement, malgré qu’elle n’ait pas de connaissances informatiques particulières.

1.4.5 Les développeurs

L’équipe de développeurs représente la plus grande partie de l’entreprise. Le programme vendu étant en perpétuelle évolution, il faut

pouvoir mettre à jour et faire avancer rapidement les différents aspects de l'application.

Tous les développeurs n'ont pas le même travail, certains sont « spécialisés » dans une tache (développement, rapports d'impression – export de fichiers, cartographie, etc.), ce qui explique la plus grande diversité de l'équipe. Néanmoins, tous les différents modules appartiennent à la même application, et chaque développeur n'est pas indépendant, toute l'équipe doit travailler ensemble et dans le même sens.

1.5 Marchés ciblés

La société développe un programme dans le monde du vin. Elle s'est donc, dans un premier temps, naturellement tournée vers un marché de proximité. A savoir : le Bordelais, le Médoc et Saint-Emilion. Qui représentent la plus grosse agglomération de grands châteaux en France.

C'est donc un marché de qualité qui va permettre de promouvoir l'image de la société. Car quand certaines des plus grandes enseignes de vin dans le monde utilisent le programme, le bénéfice est énorme pour conquérir de nouveaux marchés.

Une cinquantaine de châteaux sont à compter dans ce secteur comme clients (Dont certains châteaux de très grandes renommées : Château Yquem, Domaines Denis Dubourdieux, etc.), et une centaine est espérée pour l'année suivante.

Depuis peu, un nouveau secteur est visé : celui du Languedoc-Roussillon grâce au recrutement d'un commercial dans ce secteur. On peut donc imaginer voir de nouveaux clients dans cette région dès 2011.

Le programme est disponible en plusieurs langues : français, anglais et espagnol. Tous les textes insérés lors du développement de l'application sont ajouté dans des fichiers de langues afin d'être traduit. L'internationalisation de l'application permet d'ouvrir de nouvelles portes sur le marché mondial. En effet, le programme a été conçu de manière à pouvoir être vendu en Amérique du nord et en Amérique latine, où de nombreux châteaux importants se situent.

Pour l'avenir, tous les châteaux de vins du monde sont potentiellement des clients. De nouveaux marchés seront explorés en même temps que la société croît.

1.6 Concurrence

On peut séparer les concurrents en deux catégories. Les concurrents historiques qui vont être voués à disparaitre petit à petit car ils ont un retard technologiques trop grand, et les concurrents modernes qui se développent aussi sur la vague des nouvelles technologies.

1.6.1 Les concurrents historiques

Deux principaux acteurs sont à prendre en considération : Lamouroux et Isavigne. Ces deux sociétés possédaient des marchés depuis plusieurs années, mais leur évolution par rapport à l'évolution technologique actuelle est presque nulle. On peut donc penser qu'une

solution plus actuelle, avec tous les avantages que représentent les nouvelles technologies, va reprendre leurs parts de marché.

1.6.2 Les concurrents modernes

Ce sont les concurrents auxquels il faut faire encore plus attention. En effet, les clients cherchent des outils sophistiqués et précis afin de leur permettre de faire des mesures et d'analyser les relevés en temps réel depuis leur application. La société Neotic propose une application web possédant des sondes et des images satellites.

La seconde société est la plus importante. En effet Isagri possède plus de 800 employés et sa force commerciale est conséquente. De plus, cette société est ancienne et bien implémentée.

Pour pouvoir concurrencer et prendre des parts de marchés face à ces entreprises, il va falloir être encore plus novateurs, et répondre encore plus rapidement aux attentes des clients, qui ne cessent d'augmenter en voyant l'évolution actuelle des programmes sur ce marché.

1.7 Atouts de process2wine

Si le logiciel process2wine commence à s'imposer dans le marché vitivinicole, c'est qu'il possède des atouts novateurs et propose des solutions que les concurrents délaissent.

1.7.1 OSEO

OSEO est le nom d'une organisation publique qui finance les projets innovants. Cette organisation à trouvé le logiciel process2wine moderne et innovant, et a donc procuré une aide de 27k€ à la société D2E pour sa création.

1.7.2 Lois sanitaires

Dans le monde de l'agro-alimentaire, les lois sanitaires européennes deviennent de plus en plus strictes. Il faut savoir d'où vient le produit à l'origine, quels produits sont ajoutés, comment il a été distribué.

Le logiciel process2wine a l'avantage par rapport à ses concurrents d'être extrêmement complet pour détailler les étapes de la production, la traçabilité des produits et des lots de vins.

Quand il fallait avant reprendre des classeurs de plusieurs mois de travail pour essayer de retrouver l'origine exacte de la grappe de raisin qui fait partie d'une bouteille, le logiciel va permettre de le faire en quelques clicks, et d'imprimer toute la traçabilité dans des formats utilisables pour envoyer aux organismes de contrôles.

1.7.3 Centralisation des données

Plusieurs domaines comportent plus d'un château. Process2wine permet de centraliser les données de plusieurs propriétés dans la même application. Il va donc être bien plus facile pour les clients de gérer leurs différentes propriétés puis qu'ils auront accès à l'ensemble de leurs données sur la même page, avec la possibilité de filtrer sur l'une ou l'autre de leur propriété.

Les personnels qui travaillent dans un seul château n'auront accès qu'à leur domaine, et ainsi pour eux tout sera transparent, alors que pour l'administration, tout sera complet.

1.7.4 Entièrement en ligne

Avant, les programmes de gestion des châteaux étaient des applications lourdes. Il fallait les déployer sur tous les pc devant l'utiliser et on ne pouvait pas y accéder d'un nouveau poste ou de l'extérieur.

Grâce à une solution entièrement en ligne, process2wine est accessible de n'importe où.

C'est un avantage énorme de pouvoir entrer ses travaux directement à partir du chai, ou d'un bureau sans devoir faire la queue derrière le pc qui possède le programme.

De plus, les personnes voulant vérifier ou ajouter des informations de chez elles, peuvent désormais le faire sans devoir retourner au château.

1.7.5 Evolutif

C'est le point fort du logiciel. Process2wine a été conçu de manière à être en perpétuelle évolution, et les perspectives d'évolution du programme sont tellement riches que les projets principaux remplissent les plannings pour 3 ans.

Les programmes de gestion actuels sont très ciblés, presque aucun ne propose de suivi des clients sous forme d'ERP, de suivi des ventes, de Business Intelligence.

Toutes sortes de projets vont être rattachées à l'application dans l'avenir, et les clients peuvent déjà entrevoir les capacités du logiciel à s'adapter avec quelques uns de ceux-ci.

1.7.6 Proche des clients

La société D2E reste très proche de ses clients, et à l'écoute de leurs attentes. C'est un atout considérable d'avoir un outil qui se formate exactement à son travail, c'est pourquoi la société prend le temps de faire des écrans personnalisés pour certains clients.

Les grosses entreprises qui ont un programme rigide ne s'adaptant pas à un client particulier sont donc désavantagées vis-à-vis de la flexibilité de process2wine.

De plus, quand plusieurs écrans différents sont développés pour effectuer la même action, mais par une méthode de travail qui diffère un peu, on est presque sûr qu'une de ces méthodes pourra convenir à un nouveau client, et on aura un panel de choix à lui présenter. Ce qui est une force de vente.

2. Migration d’une application Access vers une application Web riche

2.1 Avant propos

2.1.1 Problématique

Le programme initial étant développé sous Microsoft Access, et possédant son propre fichier de base de données, il fallait trouver une solution permettant de pouvoir commencer la migration de ce programme vers une nouvelle application web, tout en permettant à l'ancienne de continuer à fonctionner.

Le choix des langages de programmation à utiliser doit être judicieux car l'évolutivité et la flexibilité sont les maitres mots de la nouvelle application. Il fallait donc étudier les différentes possibilités alliant modernité et robustesse.

De plus, le programme étant extrêmement complet, il fallait trouver un moyen de pouvoir recréer rapidement les écrans afin que la solution web soit disponible dans de brefs délais et ne demande pas des années de travail comme lors de la création du programme Access.

Enfin, il fallait trouver et savoir mettre en place des outils s'intégrant à une application web, qui soient encore plus riches que ceux proposés par l'application lourde.

2.1.2 Choix du sujet

Au cours de mon stage, j'ai pu effectuer des tâches diverses de développement pour l'application process2wine. Etant donné qu'il faut recréer et moderniser des mécanismes qui existent déjà, j'ai pu travailler sur beaucoup d'écrans différents, dont certains complètements inédits.

J’ai choisi de faire l’étude de l’étape de la migration dans ce mémoire, car c’est la partie pour laquelle j’ai été le plus libre de proposer mes réflexions et mes choix quant à la manière de procéder.

De plus, c’est la plus grosse étape de mon stage, c’est ce qui a permis à l’application de prendre forme sous sa version web. C’est donc une partie qui m’a particulièrement enthousiasmé, et sur laquelle j’ai passé du temps, à réfléchir ou à développer. C’est donc naturellement que je vais décrire ici la vision des différentes étapes de la migration que j’ai eu pendant ce projet.

2.2 Modification du programme Access

2.2.1 Le produit avant et après

Process2wine est l'évolution web du programme sous Access qui servait à gérer les châteaux de vins du groupe Taillan.

Le produit est donc à la base une application lourde, très complète car améliorée pendant plusieurs années sous forme de divers modules. Malheureusement, cette application est presque impossible à maintenir et ne permet pas une évolution suffisante pour l'avenir.

C'est le coût important pour continuer à développer ce programme qui a poussé le groupe de château à arrêter le développement de leur projet de gestion qualité.

Le programme étant vraiment complet et intéressant pour le monde du vin, il était important qu'il soit transformé. Emmanuel Thibierge et Alain Sutre l'ont compris lorsqu'ils ont racheté au groupe Taillan les droits du programme pour le transformer en un logiciel compétitif, intuitif, évolutif et surtout à la pointe des nouvelles technologies.

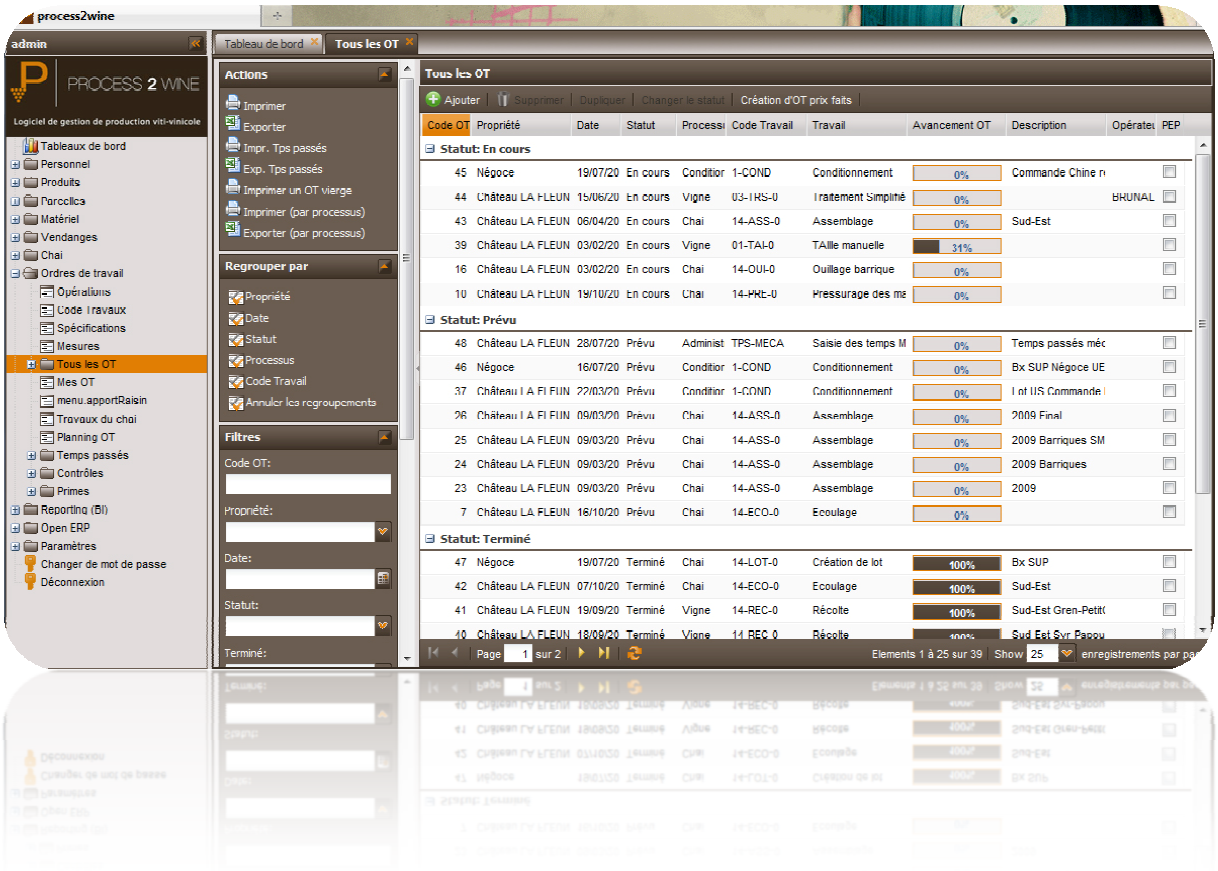
Afin d'être réellement compétitif, il fallait une application novatrice, qui présente des perspectives d'avenir et d'évolution, mais qui soit également rapidement disponible pour ne pas que la migration du programme Access se transforme en réalité en une si imposante tâche qu'elle prendrait autant de temps que de redévelopper complètement le programme sous le format web.

Le produit initial devait donc être très peu modifié pour pouvoir continuer à fonctionner pendant la période de développement des plus

gros axe de l'application web, car il était destiné à disparaitre. Tandis que l'application web devait s'enrichir très vite des fonctionnalités essentielles de l'ancien programme, pour être disponible au plus vite avec un panel d'écran indispensable à la démonstration commerciale auprès des clients.

Il fallait donc analyser les différentes solutions qui s'offraient pour chaque étape de la migration, afin de choisir celle qui convient le plus au fil conducteur du projet, à savoir : une migration qui modifie le moins possible l'ancien programme, et qui enrichie très vite le nouveau.

Ci-dessous, un écran de l'application process2wine version web, montrant une des listes de données migrée à partir d'Access.



2.2.2 Analyse du travail à effectuer

La première étape avant d'entamer la migration de l'application, était de réfléchir à la manière de procéder pour que le programme sous Access puisse continuer de fonctionner pendant toute la durée du développement web.

Chaque programme Access fonctionnait avec son propre fichier de base de données, alors que pour l'application web, nous devions utiliser un serveur de base de données.

Il fallait donc choisir le nouveau type de base de données à utiliser, recréer un schéma à partir du programme Access, transférer toutes les données et faire en sorte que le programme marche avec ce nouveau type de base de données.

2.2.3 Choix de la base de données

Pour commencer la migration des données vers un serveur de base de données, il fallait d'abord choisir quel type de base utiliser. Sur le marché, tout un tas de base de données sont disponibles et nous devions étudier laquelle serait la plus à même de fonctionner correctement avec le programme Access, et avec le programme Web tout en offrant une bonne stabilité et de bonnes performances pour le futur de l'application.

Emmanuel Thibierge ayant prévu, même avant la création de l'application web de pouvoir la relier à un système d'ERP[1] pour lequel il avait déjà effectué une formation, l'utilisation du même type de base de données qu'OpenERP (anciennement TinyERP) était un avantage énorme.

[1] Enterprise resource planning

OpenERP était relié à une base de donnée de type PostgreSQL. Je devais donc vérifier si ce type de base de données remplissait tous les prés requis pour la migration et le développement de l'application web.

Après une étude sur le site de la marque, les différents points important ont été validé :

- Open Source et gratuit : Il fallait une base de donnée gratuite, car afin d'être compétitif, cela fait un élément en moins à imposer au client. De plus, cela permet d'avoir une très grande communauté derrière le produit pour pouvoir régler les éventuels bugs très rapidement.

- Multiplateforme : Certains serveurs peuvent varier de système d'exploitation. Par exemple les développeurs sous Windows peuvent utiliser la base de données, tout comme les serveurs linux du Datacenter ou les différents systèmes chez les clients qui veulent héberger eux même entièrement l'application.

- Compatible avec Microsoft Access : très important, car il faut pouvoir se connecter à partir de l'ancien programme vers le nouveau type de base de données. Le développement d'un driver pour pouvoir gérer cette tâche aurait été une surcharge de travail importante et inutile.

- Compatible avec Hibernate : la nouvelle application web utilisant le projet Hibernate pour gérer la persistance des objets en base, il fallait également que le type de base de données soit compatible avec ce framework.

- Mis à jour régulièrement : pour commencer le développement d'une application flexible et très évolutive, il fallait une base de données qui ne soit pas laissée à l'abandon. En effet, depuis plus de 15 ans PostgreSQL n'a cessée d'évoluer et d'être amélioré. C'est un partenaire sur lequel on peut compter dans l'avenir.

- Bien documenté : Si on passe plus de temps à chercher comment faire une action qu'à la réaliser, alors on finit par perdre trop de temps. Il fallait donc un système bien documenté, avec des exemples et une communauté active. PostgreSQL répond à tous ces critères car il possède une documentation organisée qui peut être commentée par les utilisateurs.

- Outils de gestion graphiques : C'est un plus, mais non négligeable selon moi. Un programme pour réaliser les actions simples et visualiser les données rapidement est un avantage énorme. On peut vérifier en quelques clicks la cohérence de nos données, en étant certain de ce qu'on voit.

2.2.4 Amélioration du schéma de base de données

Le programme Access étant un regroupement de plusieurs modules, le schéma de la base de données a été patché et repatché. Il fallait donc le repenser, donner des noms plus explicites aux tables qui n'en avaient pas forcément. Renommer également certains champs et faire en sorte de ne pas avoir à changer toutes les requêtes du programme Access utilisant les anciens noms.

De plus, l'application comporte plus de 100 tables, il était donc impensable de recréer le schéma à la main.

Il fallait donc étudier la meilleure manière de procéder, et j'en suis venu à la conclusion que le développement d'une application en VBA pour boucler sur le schéma Access était le plus avantageux.

Ainsi, une fois le programme créé, il ne nous restait plus qu'à définir la syntaxe du fichier de sortie. En effet, on pouvait, pour chaque table et

chaque champ, ajouter une ligne dans un fichier, ce qui nous permettait de pouvoir manipuler le schéma extrêmement librement, et organiser les informations en fonction de nos besoins.

Etant donné qu'il fallait renommer beaucoup de tables et de champs, et que seul Emmanuel Thibierge pouvait le faire, car il fallait que le schéma corresponde à sa vision métier de la futur application ; le premier export du schéma grâce à ce programme a été un fichier CSV, permettant d'être importé sous Excel et de pouvoir voir facilement les différentes tables et leurs champs, pour leur associer un nouveau nom de correspondance.

Une fois le fichier complété par Emmanuel Thibierge avec les correspondances de chaque table et chaque champ, on pouvait attaquer l'étape suivant : la migration de la base avec la création du schéma et l'import des données.

2.2.5 Création de la nouvelle base

Grâce aux étapes précédentes, nous disposions de plusieurs outils différents afin d'entamer le processus de création de la base de donnée :

- Le programme en VBA qui permet de boucler sur l'ancien schéma Access et pouvoir ainsi générer dans un fichier des données se rapportant à ces données.

- Le fichier de correspondance entre les anciens noms de tables et de champs, et les nouveaux noms qui doivent les remplacer lors de la création du nouveau schéma.

Ici, les méthodes envisageables pour recréer le nouveau schéma étaient très limitées, car nous devions le générer à partir du schéma Access, et également remplacer certains noms de champs et de tables.

Il y avait toujours la solution de le recréer à la main, mais cette solution est, comme précédemment, très vite écartée car beaucoup trop longue et des outils modernes nous permettent d'éviter cette perte de temps inutile.

J'ai donc préféré m'axer sur un mécanisme tout automatisé, qui aurait pour base le programme VBA qui boucle sur le schéma Access. Le principe était simple : il fallait boucler sur le schéma et générer dynamiquement dans un fichier une requête SQL de création de table qui comprendrait tous les champs.

Mais une contrainte s'ajoutait. Il fallait échanger des noms de tables et de champs par de nouveaux noms, mais il fallait également que le programme Access continue de fonctionner en étant branché sur une nouvelle base de données qui comporte des tables avec de nouveaux noms.

Certains écrans de l'application comportent des dizaines de requêtes et la modification de tous les noms de champs dans le logiciel était bien trop longue. Il fallait donc trouver une solution permettant de garder un programme qui marche, et également de générer le nouveau schéma avec les nouveaux noms.

La meilleure méthode était donc d'ajouter au programme VBA une fonction qui nous permette d'ouvrir le fichier CSV qui associe un nouveau nom de champs à l'ancien. Puis de boucler sur le schéma Access et échanger les noms de champs par les nouveaux en comparant le champ actif avec le fichier de correspondance des noms.

Et enfin générer un script dans un fichier, qui permettra de créer le schéma quand exécuté.

Etant donné que nous générons dans un fichier des scripts SQL, il était alors facile d'ajouter des lignes supplémentaires pour créer des vues SQL qui vont faire la jointure entre les anciens et les nouveaux noms de champs.

Le mécanisme était alors complet :

- Il fallait créer une fonction pour ouvrir le fichier de correspondance des noms de tables et champs sous Access.

- Ensuite il fallait utiliser l'ancien programme pour boucler sur le schéma, mais en appelant cette fois une fonction qui génère des scripts SQL dans un fichier, tout en comparant les noms des champs qui sont bouclés pour trouver leur correspondance dans le fichier précédant.

- Il fallait ajouter pour chaque table, un script de création d'une vue qui permet de garder l'association des anciens noms vers les nouveaux et permette à l'ancien programme de continuer à fonctionner

Une fois le programme assemblé comme il faut, nous n'avions donc plus qu'à exécuter le fichier dans lequel les scripts SQL ont été générés, et le nouveau schéma était créé tout en gardant l'ancien programme fonctionnel pour la durée du développement de l'application Web.

2.3 Choix des outils pour le développement de l'application web

Une fois le programme Access fonctionnel en étant relié à la nouvelle forme de base de données, il fallait réfléchir à commencer développer une application web qui ne procure pas seulement un avantage de portabilité, mais qui puisse enrichir l'application et lui donner de nouvelles perspectives.

Pour cela, il fallait étudier quels outils étaient les plus propices à la réalisation de ce projet, quelles technologies utiliser par rapport à d'autres pour être à la pointe de la technologie mais ne pas tomber dans les nouveautés qui ne durent pas.

Et surtout pouvoir réaliser rapidement et à moindre coûts un outil stable et robuste en lequel les clients puissent avoir entièrement confiance.

2.3.1 Pré requis

Le choix des bons outils est primordial pour l'avenir de l'application, car quand elle aura quelques années ou même quelques mois de développement, il sera presque impossible de changer toute la structure du logiciel pour changer les outils choisis au départ.

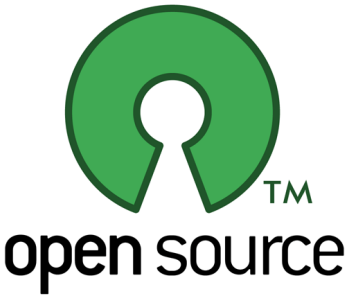
Il fallait donc étudier le panel de possibilités qui se présentaient à nous, et les comparer sur plusieurs critères :

- Confiance dans l'éditeur de l'outil
- Coût de la mise en place
- Evolutivité
- Fréquence des mises à jour
- Documentation
- Robustesse

Par exemple, changer le langage de programmation d'une application reviendrait à devoir la recréer entièrement. Il fallait donc s'assurer de faire les bons choix techniques, mais aussi stratégiques.

2.3.2 Open Source et Free Software

C'est le premier choix stratégique qui a été adopté. Afin de ne pas augmenter les coûts de développement en utilisant des outils hors de prix, pour lesquels il faut des licences pour chaque ordinateur utilisant le programme, le choix d'utiliser de logiciels gratuits et open source a été rapidement adopté.



De plus, ce choix permet, au niveau marketing, d'être toujours plus compétitif. Car plus les sommes investies dans les outils de développement sont grandes, plus le prix de revient du produit augmente. Tandis qu'utiliser des outils gratuits permet de ne pas augmenter le coût de revient du produit, et ainsi proposer un logiciel de la même qualité à un prix beaucoup plus compétitif vis-à-vis des concurrents sur le marché.

Un logiciel Open Source est, selon l'Open Source Initiative, un logiciel avec la possibilité de libre redistribution, de libre accès au code source et de travaux dérivés. Attention, car toutes fois, un logiciel open source gratuit, peut être modifié et vendu. C'est ce que l'on appelle une distribution.

Les avantages des logiciels open source se situent souvent dans leur communauté. Puisque tout le monde a accès au code source, tout le monde est à même de pouvoir modifier le programme et de partager gratuitement ou non ses améliorations avec le reste de la communauté.

De plus, chacun peut retoucher le code source, et donc améliorer le programme là ou un seul programmeur n'aurait pas forcément pensé à un certain type d'optimisation.

Il s'avère aussi essentiel d'avoir une grande communauté pour tester le logiciel. Toute application comporte certains bugs, parfois difficilement décelables et plus on va avoir de personnes qui utilisent et testent l'application, plus nous allons avoir un retour important sur les possibles bugs et les corrections à apporter.

Pour une application telle que process2wine, il était presque indispensable d'utiliser des librairies et des langages open source, car il est souvent inutile de réinventer la roue. Et quand il y a une énorme communauté derrière un logiciel libre, alors on peut être presque sûr que quelqu'un a déjà pensé ou réalisé ce que l'on veut faire. Cela permet de trouver facilement des outils qu'on aurait mis du temps à développer, ou de l'aide concernant un sujet un peu précis que seules certaines personnes travaillant dans le même domaine peuvent connaitre.

Heureusement, grâce à l'essor important qu'à vécu Internet, les projets Open Source fleurissent grandement à notre époque et sont maintenant très bien encadrés par des équipes de professionnels.

Il est donc d'un coté plus facile de trouver plusieurs outils correspondant à nos besoins, mais aussi plus difficile de choisir celui qui nous correspondra le mieux sur le long terme. Il a donc fallu, pour chaque outil, faire une petite étude de marché et comparer les différents concurrents afin de trouver le produit le plus adapté au développement de notre application Web.

2.3.3 Le choix du langage de programmation

Après une petite étude, 3 langages sortent du lot et méritent qu'on s'attarde à les comparer pour savoir lequel choisir :

- PHP[2] (PHP5)
- Java (J2EE)
- .NET (ASP.NET)

Comme spécifié dans l'article du dessus, nous voulions nous orienter vers des langages gratuits, multi plateformes et robustes.

On a donc pu éliminer le langage Microsoft .NET qui n'est pas open source et qui ne peut fonctionner que sur des machines Windows (surcoût par rapport à des distributions Linux gratuites), malgré qu'il possède un logiciel de développement qui est à mon sens le meilleur du marché (Visual Studio), mais qui fait lui aussi parti d'un surcoût pour la société comparé aux autres éditeurs open sources pour Java et PHP.

De plus ce langage possède une communauté moins importante que Java ou PHP étant donné qu'ils sont open source, plus de personnes peuvent développer sur ces langages.

Il nous restait donc à choisir entre PHP et Java. Voici un petit comparatif des deux langages :

PHP	Java
Très souple	Très rigide
Léger mais se charge à chaque fois	Lourd mais ne se charge qu'une fois
Presque pas de débogage possible	Débogage poussé grâce aux IDE
Modèle objet encore jeune	Modèle objet très complet

[2] PHP : Hypertext Preprocessor

Peu gourmand en ressources	Très gourmand en ressources
Documentation avec exemples et ouverte aux commentaires	Documentation sans exemple, gérée par Sun et fermée aux commentaires
Pas de méthode de gestion des fichiers particulières	Promotion de la séparation de l'interface, de la logique et des données (MVC)
Multiplateforme	Multiplateforme
Open source	Open source
Langage interprété	Langage compilé
Possède encore peu de Framework pour faciliter la gestion de certaines tâches	Possède de bons outils pour faciliter certaines tâches

Lors de mes recherches sur les points qui associent ou différencient ces deux programmes, j'ai souvent trouvé comme conclusion que les auteurs des articles n'utiliseraient pas PHP pour bâtir une application d'entreprise complète qu'il faut maintenir et faire évoluer.

Je suis arrivé à la même conclusion, car si on étudie le petit tableau ci-dessus, on peut voir que les prés requis principaux sont validés par les deux langages :

- Open source
- Gratuit
- Multiplateforme
- Grande communauté

Même si le point sur la documentation est marqué par PHP, la communauté Java est assez grande pour trouver tout ce dont on a

besoin, dont les commentaires des utilisateurs sur d'autres sites que le site officiel de SUN.

En regardant d'un peu plus près, on voit que PHP est un langage interprété. Pour une grosse application, il est nécessaire d'avoir un outil qui va compiler l'application régulièrement pour vérifier s'il n'y a pas d'erreurs qui se sont glissées dans le code, et ne pas déployer des fichiers non fonctionnels. C'est donc le premier point qui a fait pencher la balance vers Java.

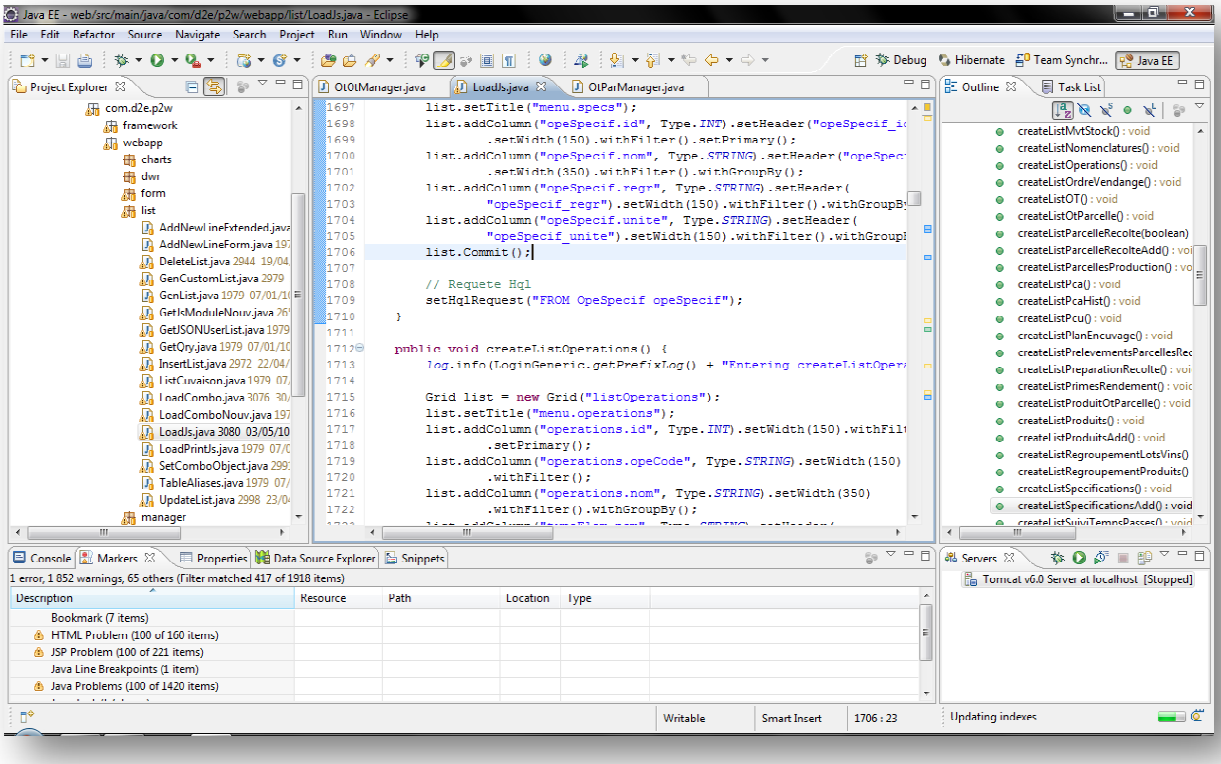
Ensuite, la gestion des fichiers Java est bien plus rigide, ce qui est un avantage pour le maintient de l'application. Les imports de classes qui n'existent plus vont provoquer des erreurs de compilation et seront donc repéré instantanément, les fichiers sont ordonnées sous forme de packages et non de dossiers ce qui permet d'avoir une organisation plus stricte lors des imports, et les outils de développement sont bien plus riches que ceux proposés pour PHP (souvent plus accès sur l'élaboration du design, ou alors se rapprochant d'un bloc-notes en couleur).

Enfin, l'absence de moyens pour déboguer sérieusement une application PHP a permis de confirmer l'utilisation du langage Java. Car lorsqu'une erreur survient en PHP, le script s'arrête, et rien ne nous permet de le savoir. Tandis que Java va générer des exceptions (les exceptions sont des rapports d'erreurs brutes) qui seront affichées dans la console de logs du programme.

Pour une application qui doit être robuste, maintenable, et évolutive, on ne pouvait pas se permettre de choisir un langage qui est plus facile à mettre en place, qui donnera peut être des résultats plus rapidement mais qui va devenir plus compliqué à déboguer méthodiquement et à structurer pour une évolution constante sur le long terme.

De plus, Java possède des outils qui permettront de gérer la persistance en base de données comme Hibernate, des outils pour gérer la compilation constante des fichiers et indiquer s'il y a des erreurs, et beaucoup d'autres outils qui vont se greffer à l'IDE[3] et permettre un développement efficace et structuré de l'application par toute l'équipe en collaboration.

Ci-dessous l'IDE Eclipse pour développer sous Java



Java étant un langage coté serveur, il nous fallait maintenant trouver le moyen de rendre notre rendu HTML[4] riche.

[3] Integrated Development Environment
[4] Hypertext Markup Language

2.3.4 Le JavaScript

Attention de ne pas confondre le JavaScript et le Java car se sont deux langages totalement différents.

Le JavaScript est un langage en plein essor qui suscite depuis quelques années un énorme intérêt car il permet de créer toutes sortes d'améliorations visuelles et pratiques pour un internaute.

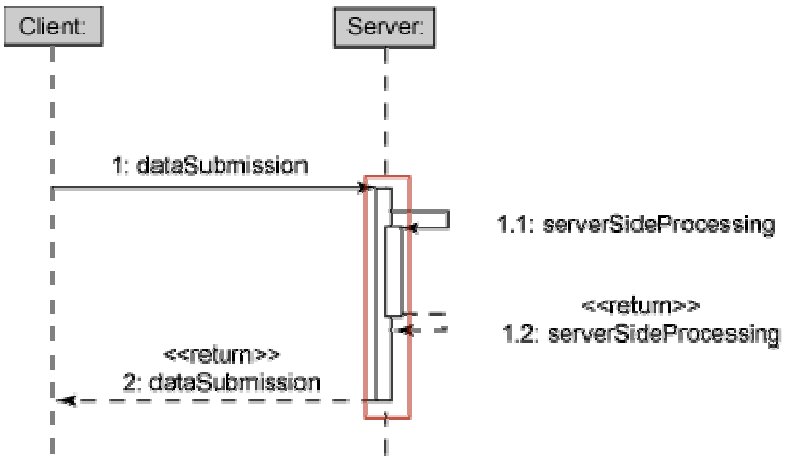
Pourquoi se tourner vers ce langage ? Ce langage nous a intéressés car nous avions la couche serveur de notre application, et qu'il nous fallait donc créer et afficher un site web qui permette à l'utilisateur de vivre une expérience riche et ne pas rester dans le site web basique.

En effet, lorsque l'on navigue sur un site web, on est souvent confronté au fait qu'il faut attendre que les pages se chargent et qu'on ne peut rien faire pendant ce temps.

C'est pourquoi le langage JavaScript nous a intéressé, car depuis peu, internet est en train de se tourner vers la gestion asynchrone des requêtes, ce qui permet de ne plus devoir attendre patiemment derrière son écran en attendant une réponse du serveur avant de pouvoir continuer à naviguer.

Comment cela fonctionne :

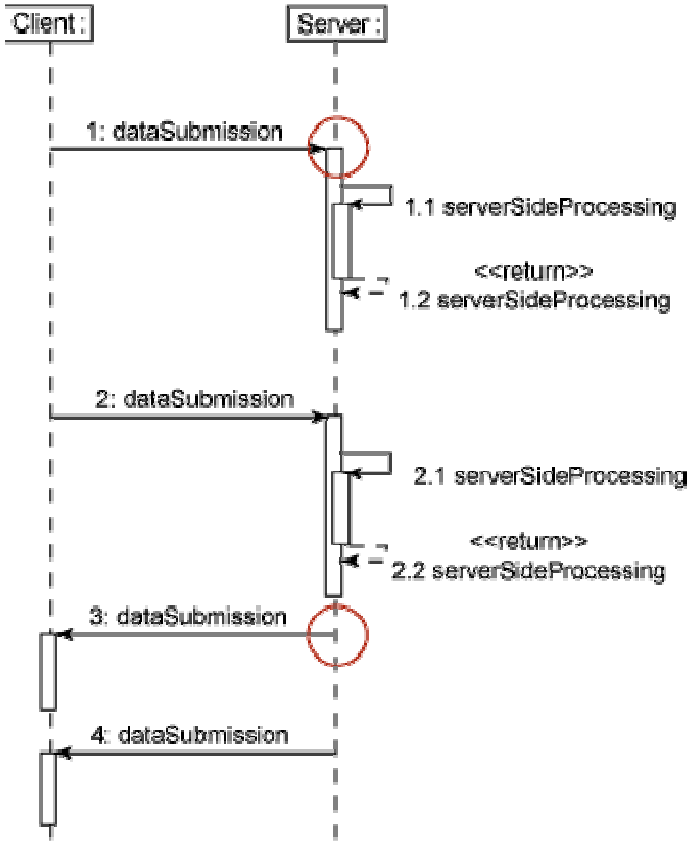
- Une requête synchrone suit ce schéma :



Le client clique sur un lien, une requête est envoyée au serveur qui va la traiter, puis renvoyer une réponse au client qui sera redirigé vers la page désirée.

Malheureusement le client devra attendre tout le temps pendant lequel la requête est traitée par le serveur avant de pouvoir continuer ce qu'il faisait.

- Si maintenant on regarde du coté d'une requête asynchrone :



On peut voir que le client va envoyer plusieurs requêtes qui ne seront pas bloquantes pour lui. Par exemple, si il décide d'appuyer sur le bouton sauvegarder d'une application web, puis qu'il veut ouvrir un nouveau formulaire, alors les données à enregistrer seront envoyées au serveur qui va traiter la requête de son coté, tout en laissant l'utilisateur libre de ses mouvements.

Quand les actions de traitement coté serveurs sont terminées, alors la requête retourne un statut ou des informations qui seront interprétées par le site web seulement au moment de leur retour, et il ne sera pas bloqué en attendant de pouvoir traiter la réponse comme lors des requêtes synchrones.

Il était très important pour nous de trouver un moyen de faire bénéficier à l'utilisateur d'une ergonomie et d'une fluidité d'utilisation de l'application qui lui permette de ne pas être frustré de toujours devoir attendre après le logiciel. Lorsque l'on développe une application très complète, qui sera de plus en plus enrichie au fil du temps, il fallait trouver un bon moyen pour la rendre aussi agréable à utiliser qu'une application lourde (programme exécutable).

Ce qui nous amène à la seconde grande amélioration que peut apporter le JavaScript à notre application : des outils très évolués comme des arbres hiérarchiques, des listes de données que l'ont peut trier, et énormément de composants très riches.

De plus, grâce à la technologie de requêtes asynchrones AJAX[5], et à la possibilité de lancer et écouter des évènements provoqués par l'utilisateur en JavaScript, on peut imaginer toutes sortes de possibilités pour rendre notre application beaucoup plus riche.

Etant donné que le JavaScript devient incontournable, plusieurs Framework se développent et permettent de gérer plus facilement certaines actions.

Il nous fallait donc trouver un Framework JavaScript qui soit vraiment très complet, sûr, évolutif et qui propose des outils graphiques permettant de recréer un environnement agréable et familier pour l'utilisateur final.

[5] Asynchronous JavaScript and XML

2.3.5 Le choix d'un Framework Graphique

Afin de disposer d'outils graphiques poussés, il fallait trouver un Framework qui nous permette de dessiner rapidement une application web riche, grâce à un panel de composants faciles à mettre en place.

Il fallait également que ce Framework fonctionne sur tous les navigateurs, ce qui est un peu plus compliqué étant donné que certains navigateurs ne respectent pas les normes JavaScript (notamment Internet Explorer qui a préféré inventer sa propre norme avec le JScript).

Après quelques recherches, on s'aperçoit qu'il y a un petit choix possible entre :

- ExtJS
- Yahoo UI
- OpenFaces
- Ajax.org
- Google Web Toolkit
- SmartClient

Le choix s'est rapidement restreint entre deux principaux Framework, ExtJS et Yahoo UI, qui sont tous les deux fonctionnels sur tous les navigateurs, et qui possèdent le plus large panel de composants.

Il faut savoir qu'ExtJS est une adaptation des fonctionnalités que propose YUI[6] remaniée par une équipe d'experts en JavaScript. On peut donc être sûr que toutes les fonctionnalités de YUI seront reprises par ExtJS, ce qui n'est pas vrai dans le sens inverse.

De plus, les composants d'ExtJS sont beaucoup plus soignés graphiquement, ce qui nous a poussés à nous tourner vers ce Framework.

[6] Yahoo User Interface (Yahoo UI)

D’après la documentation sur le site officiel d’ExtJS, qui a été récemment renommé en Sencha, nous pouvons voir qu’il est compatible avec tous les navigateurs qui nous intéressent :

- Internet Explorer 6+
- FireFox 1.5+ (PC, Mac)
- Safari 3+
- Chrome 3+
- Opera 9+ (PC, Mac)

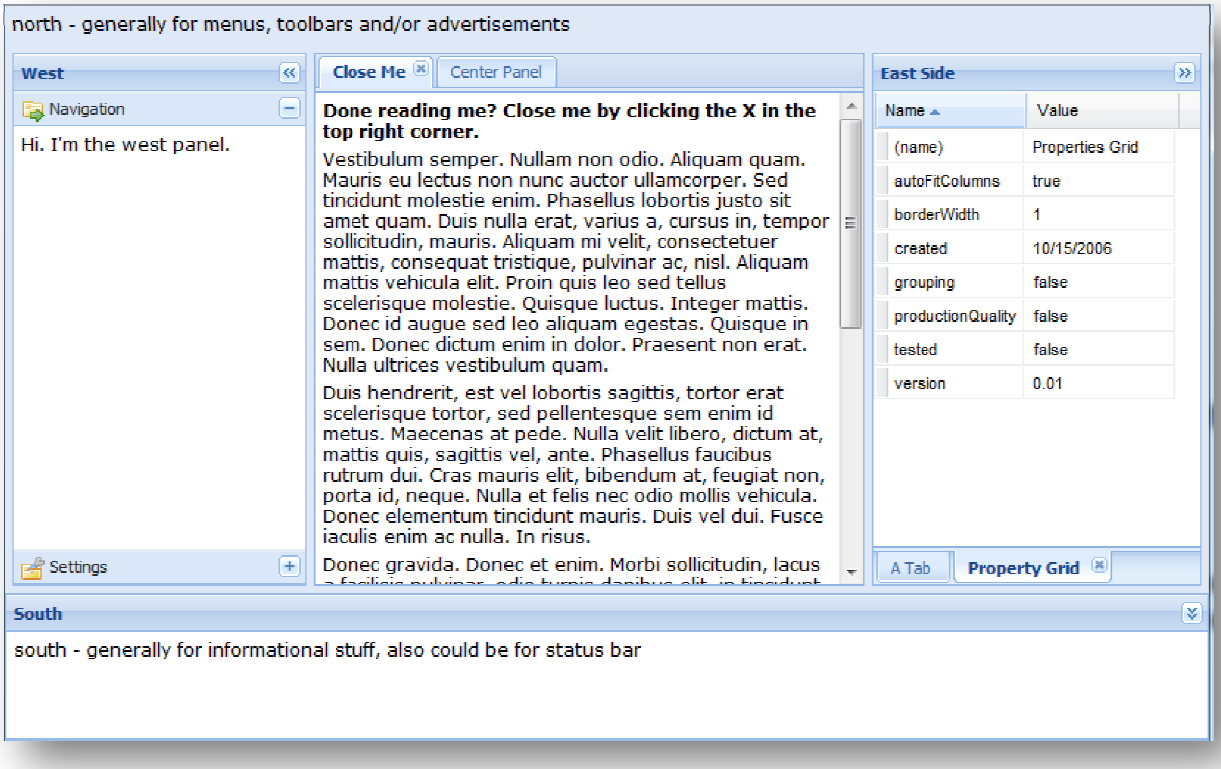
Le Framework est également Open Source, ce qui nous permet de pouvoir modifier n’importe quel composant facilement, et de pouvoir faire des modifications dans le code si nous en avions le besoin.

Le code est très structuré et hiérarchisé, ce qui permet de créer facilement des composants qui peuvent hériter d’un composant qui nous plait, afin de le paramétrer ou le modifier comme il nous convient et lui donner un nom par lequel on pourra l’appeler à chaque fois que nous en avons besoin sans avoir à le redéfinir.

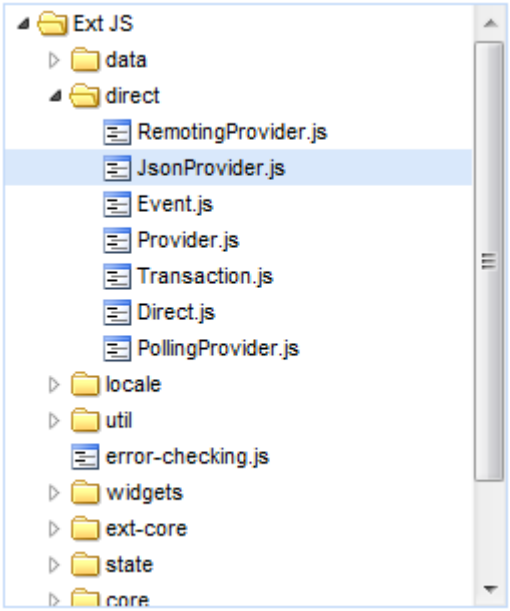
Le panel de composants graphiques mis à disposition est vraiment très bien réalisé et très diversifié. Il fallait qu’on puisse mettre en forme rapidement notre application, avec des composants riches rappelant les applications lourdes qu’on peut trouver sous Windows ou autre de nos jours.

Voici quelques exemples de composants dont nous avions absolument besoin pour notre application Web :

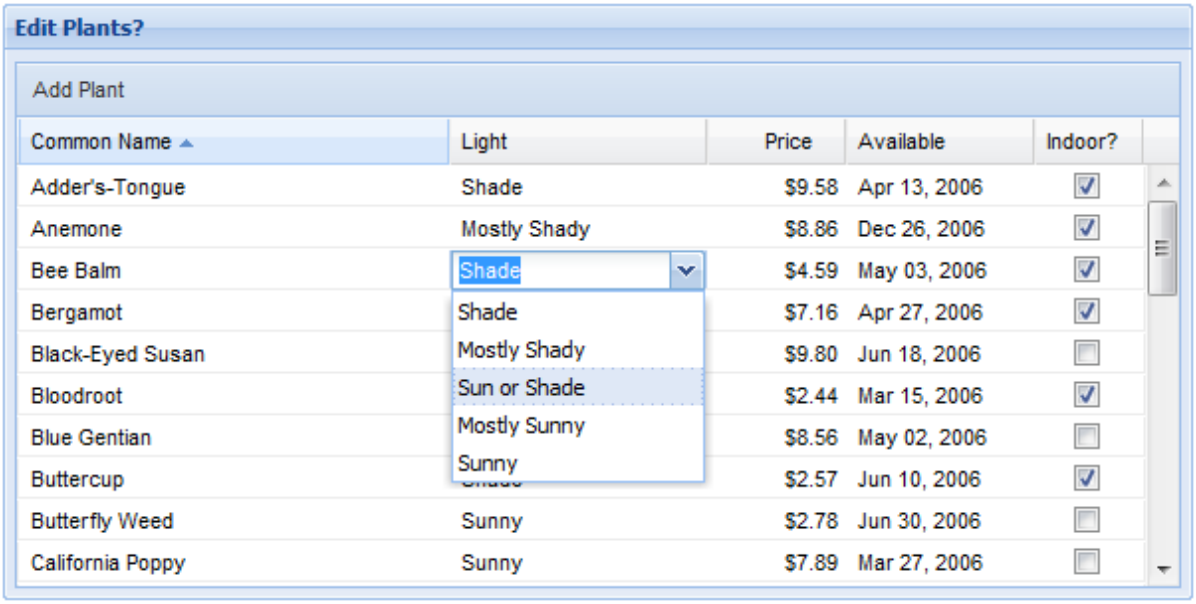
- Un Layout Manager, qui va permettre de gérer très simplement la mise en forme de l’application en plusieurs panneaux :



- Un composant pour créer des arbres hiérarchiques, afin de pouvoir ordonner un menu très complet, ou différentes informations à regrouper par dossiers :



- Et enfin, un composant permettant de lister sous forme de tableaux toutes sortes d'informations. C'est le composant principal qu'il nous fallait pour la migration du programme Access, afin que les listes puissent être rapidement migrées et que le changement de technologique apporte des bénéfices bien réels.

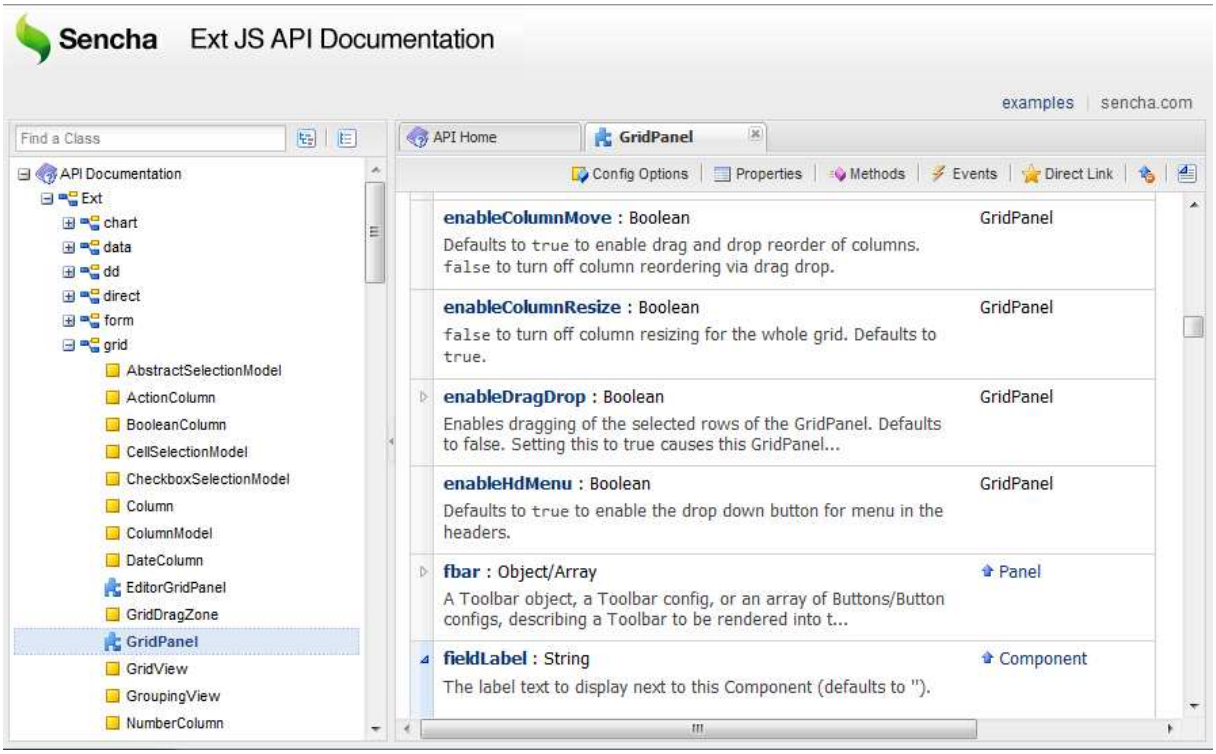


Grâce à des composants très bien réalisés, extrêmement paramétrables et qui évoluent sans cesse, nous avions tous les outils nécessaires au démarrage de l'application web.

Il faut également noter qu'ExtJS propose une documentation très bien réalisée, grâce à leur Framework, donc entièrement dynamique. Cette documentation est complète, elle spécifie toutes les informations nécessaires aux composants, ainsi que les paramètres hérités des classes mères. Ce qui permet d'avoir en un coup d'œil, toutes les options possibles pour paramétrer un composant.

De plus, lorsqu'on a besoin de se rapporter à un autre composant pour plus d'information dans une fonction, un lien nous ouvre un nouvel onglet vers le composants, fait défiler la page jusqu'à l'endroit qui nous intéresse, ce qui permet de gagner énormément de temps et d'apprendre extrêmement vite ce langage.

Voici à quoi ressemble la documentation :



La communauté derrière ExtJS est également vraiment importante et active. C'est ce qui démarque encore plus ce Framework de ses concurrents. Quand on va sur le forum de Sencha, il y a sans cesses de nouveaux sujets, et l'équipe de développement elle-même prend le temps de dépondre aux utilisateurs quand ils ont des problèmes.

Pour avoir discuté quelques fois sur ce forum, lors de l'apprentissage du Framework, où même plus tard en ayant un bon niveau, on se rend compte que la communauté aide beaucoup à faire avancer le projet, alors que certains Framework peuvent être décourageant car on a l'impression que personne ne peut nous soutenir si on rencontre des problèmes.

Des versions d'ExtJS paraissent régulièrement, alliant certains patchs pour corriger les quelques bugs, à toujours plus de nouveaux composants intéressant. On peut donc miser sur cette solution pour le

futur de notre logiciel et penser qu'elle deviendra un incontournable des RIA[7] dans un avenir proche.

Enfin, comme dans tous les gros projets Open Source bien développés, la communauté fait paraître de nombreux plugins très utiles que les développeurs du Framework prennent le temps d'intégrer à ExtJS quand ils deviennent vraiment incontournables. Cela permet d'avoir, en plus de tous les composants d'ExtJS, un nombre encore plus grand de composants créés par les utilisateurs qui répondent à quasiment tous les besoins que l'on pourrait avoir dans le développement d'une application web riche.

Tous ces avantages nous ont permis de choisir ExtJS comme Framework principal de l'aspect visuel de notre application, et d'étudier les méthodes qui nous permettront de migrer le programme Access vers notre nouvelle application web en Java/ExtJS.

Nous allons donc étudier dans la prochaine partie, la réflexion sur la méthode pour générer rapidement des listes migrées à partir du programme Access vers la nouvelle application Web, en utilisant les outils que nous avons pris le temps de choisir en amont.

[7] Rich Internet Application

2.4 Génération automatique des listes avec ExtJS

2.4.1 Etude des différentes possibilités

Pour commencer, il fallait étudier les différentes possibilités de générer des listes de type « grid » pour migrer rapidement les écrans du programme Access vers la nouvelle application web.

Après réflexion, trois méthodes s'offraient à nous, dont deux méritaient d'être bien étudiées pour être certain de choisir celle qui permettra à l'application de se développer dans de bonnes conditions sur le long terme.

- La première possibilité était la plus basique : créer un fichier JavaScript par liste.

C'est la méthode de base enseignée dans les tutoriels du Framework ExtJS, mais on s'aperçoit rapidement qu'elle devient très couteuse en temps, et en ressource.

Puisque toutes nos listes sont différentes, il aurait fallu créer pour chaque liste un fichier JavaScript complètement différent, au niveau du nombre de colonnes, de la requête pour récupérer les données, pour les différents filtres à activer ou non, et pour les droits de lecture et écriture de chaque utilisateur.

C'est une méthode qui doit être utilisée pour des écrans particuliers, qui ne peuvent pas être factorisé en un mécanisme plus générique, car étant donné le nombre de listes à migrer, il était bien plus avantageux de créer un système pour automatiser le processus de création de ces listes de manière automatique.

- La seconde possibilité était de créer un composant dans la librairie ExtJS, qui hérite de la classe Grid qui permet de faire les listes.

Grâce à cette méthode, nous pouvions prédéfinir un composant générique en JavaScript, qui pourrait générer automatiquement les colonnes en fonctions du tableau de données retourné par le serveur.

On n'aurait alors plus qu'à appeler ce nouveau composant dans un fichier JavaScript, en changeant simplement le paramétrage de certaines options lors de la création de notre composant, telle que l'adresse de la requête pour charger les données, et quelques autres informations primordiales.

Malheureusement, cette méthode s'avère être une bonne manière de développer dans ce Framework, mais risque de très vite augmenter de manière considérable le temps de chargement de l'application chez le client.

Chaque fichier JavaScript doit être inclus dans le code source de la page, et donc, si chaque liste possède un fichier qui contient la définition du composant qu'elle doit afficher, nous allons devoir inclure autant de fichiers qu'il y a de listes au chargement de l'application.

Cette méthode ne s'avère donc pas assez générique, et beaucoup trop lourde sur le long terme pour le futur de notre application. Il fallait donc trouver une manière d'automatiser le processus sans pour autant alourdir le logiciel.

- La troisième possibilité était de créer un seul fichier de définition de nos liste coté serveur en Java, et de générer automatiquement le code JavaScript requis en fonction des options spécifiées.

Avec cette méthode, nous aurions une classe Java qui représenterait notre composant de Liste, dans laquelle on pourrait ajouter des fonctions génériques de personnalisation qui pourraient être renseignées différemment pour chacune de nos listes.

Il nous faudrait ensuite un fichier où on créerait la définition de toutes les listes de l'application, en appelant notre classe précédemment créée avec différentes options.

Puis, ce fichier de définition serait traité et les options transformées en code JavaScript que l'on pourrait insérer dans un fichier générique de manière dynamique grâce à l'utilisation d'une JSP[8] et des Java Bean.

Le client n'aurait donc aucun fichier à charger, car le fichier serait créé et renvoyé par le serveur seulement lors du chargement de la liste. De plus, il serait extrêmement facile d'enrichir notre classe qui représente le composant de liste et de pouvoir rajouter des fonctionnalités futures à toutes les listes de l'application en ayant centralisé le traitement.

Cette méthode était la plus pertinente à retenir dans le cadre de ce projet. Car elle répondait à tous les critères requis pour le développement de l'application :

- Evolutivité
- Facilement maintenable
- Peu couteuse en ressource pour le client
- Très grande rapidité pour recréer des listes

Maintenant la méthode appropriée trouvée, il fallait réfléchir à comment mettre en place le mécanisme. Tout d'abord, il fallait commencer par créer la classe représentant notre composant de liste.

[8] Java Server Page

2.4.2 Création d'un composant représentant une liste

C'est la classe qui devra représenter notre composant de liste. Il faut donc la penser de la manière la plus modulable et évolutive possible, car le projet ExtJS étant en constante évolution, il faut qu'on puisse rajouter dans cette classe des valeurs par défaut ou de nouvelles options de configuration rapidement. En un mot, elle doit être évolutive.

Il a tout d'abord fallu déterminer les options qui seront ajoutées par défaut à toutes les listes, et les options qui doivent être modifiable rapidement. Pour cela, il suffisait de comparer deux listes et d'identifier leurs différences.

Voici les éléments qui varient d'une liste à l'autre que nous avons pu identifier :

- Les données affichées
- Le nombre de colonnes
- Les différents champs pour filtrer
- Les boutons d'actions sur la liste courante
- La méthode d'ajout des données dans la liste
- L'alignement dans certaines colonnes
- La possibilité d'éditer directement certaines colonnes
- L'action lancée lorsqu'on modifie des données
- La possibilité de grouper par certains champs
- L'ajout ou non d'une ligne de totaux en bas de la liste
- La possibilité de masquer ou non une colonne

En partant avec ces éléments principaux comme objectifs, car d'autres viendront se greffer au fur et à mesure de l'évolution de l'application, nous pouvions commencer à créer notre composant en Java, en gardant à l'esprit que le but était de pouvoir recréer des listes très rapidement.

Etant donné que beaucoup de propriétés se rapportent à une colonne, il a tout d'abord fallu créer un objet représentant notre colonne, dans lequel nous avons ajouté les propriétés principales que l'on pourra modifier grâce aux getters/setters comme :

- Le titre de la colonne
- Le type de données qu'elle comporte (caractères, nombres, etc.)
- L'alignement des données
- Si elle comporte un filtre
- Si on doit pouvoir grouper sur les données de cette colonne
- La méthode de calcule des totaux (somme, moyenne, etc.)
- Si elle est éditable
- Si on peut la cacher

Une fois l'objet représentant notre colonne défini, on pouvait créer notre classe Grid, en créant tout d'abord un tableau de colonnes, que nous pourrons enrichir grâce à des fonctions.

Chaque paramètre qui devait être modifiable dans la liste a donc été ajouté sous forme de fonction, qui se rapporte soit à une colonne, soit à l'ensemble de la liste.

Et pour plus de facilité, chaque fonction retourne un pointeur vers notre objet de liste en cours, ce qui va nous permettre de pouvoir enchainer les méthodes de paramétrages les unes à la suite des autres lors de la définition de notre liste. Par exemple, cela permettra une plus grande visibilité de se faire se succéder sur la même ligne les propriétés en rapport, et cela permettra également de pouvoir spécifier des propriétés sur une colonne juste après sa création, sans avoir à faire une recherche dans toutes les colonnes pour trouver celle qui nous intéresse. Mais nous verrons plus en détail cette méthode de définition lors de l'étude du fichier de définition des différentes listes.

Bien entendu, chaque fonction devait être documentée via la Java Doc, afin que quand le développeur déclare son objet Grid, il puisse connaître l'action que va effectuer chaque méthode de la classe, et qu'une indication l'aide pour spécifier chaque paramètre requis.

Ci-dessous, un exemple du code de la classe Grid :

```
/**
 * Class Grid
 * @author a.sassoust
 * Permet de créer des listes dynamiquement
 */
protected class Grid {
	…

	/**
	 * Constructeur Grid
	 * @param name Nom unique de la liste,
	 * permet de créer un namespace unique pour éviter les conflicts
	 */
	public Grid(String name) {
		// Remise à zéro de toutes les variables de LoadJsGeneric
		RAZFields();

		// Initialisation des tableaux
		atomChamps = new ArrayList<AtomChamp>();
		hiddenFields = new ArrayList<AtomChamp>();
		filterFields = new ArrayList<HashMap<String, String>>();
		lstGroupBy = new ArrayList<HashMap<String,String>>();
		actions = new ArrayList<HashMap<String, String>>();
		functions = new HashMap<String, String>();
		methods = new HashMap<String, Method>();
		aliases = new HashMap<String, String>();
		buttonactions = new ArrayList<HashMap<String, String>>();

		// Valeur par défaut de certaines propriétés
		setListId(name);
		setDateFormatRender("d/m/Y");
		setDateFormatReader("Y-m-d H:i:s");
		genActions();
		genFunctions();
		setPrimaryKey("");
		genUniqueId();
		setExpandMode(Mode.AUTO);
		setAfterEdit("handleEdit");
		setLineLimit(25);
	}
	…
}
```

Maintenant que nous avions définit une classe représentant notre composant, il fallait passer à l'étape suivante : créer le fichier de définition des listes afin de commencer à migrer rapidement toutes les listes du programme Access.

2.4.3 Le fichier de définition des listes

Cette classe est la classe qui sera appelée lorsque l'on clique sur un menu qui ouvre une liste. Elle va contenir la définition de toutes les listes de l'application et charger les propriétés de celle que nous voulons afficher.

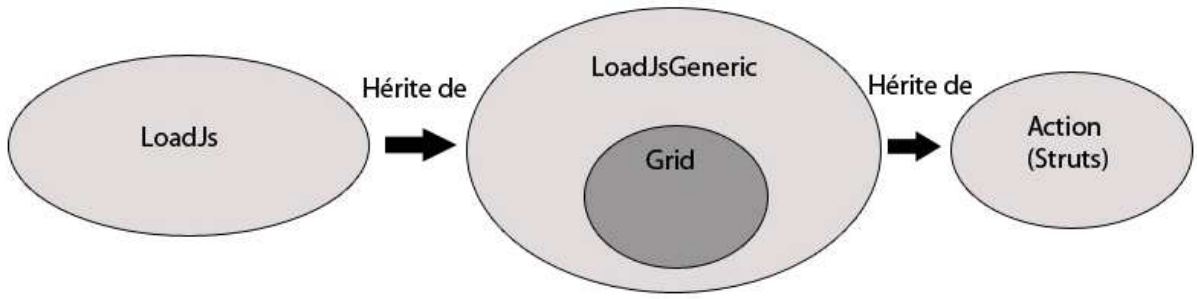
Comme notre classe composant est déjà créée, et qu'on l'a dessinée de manière à pouvoir créer des listes très rapidement, notre classe de définition des listes va être très rapide à mettre en place.

Tout d'abord, notre classe doit pouvoir être appelée lorsqu'on clique sur un menu, donc par une requête HTTP[9]. Notre application Java utilisant Struts, nous devions faire hériter notre classe de la classe « Action ».

Mais il fallait prévoir que la définition de notre composant doit ensuite être transformée en fichier JavaScript avant d'être renvoyée à l'écran de l'utilisateur, et il était donc préférable d'étudier la meilleure manière de faire le lien entre ces différentes classes.

Etant donné qu'une classe devait générer un fichier à partir de la définition de notre classe LoadJS tout en reprenant les infos stockées dans notre classe composant Grid, la meilleure solution était d'inclure la classe Grid dans la classe qui va générer le fichier, et de faire hériter notre classe de définition des données de cette classe qu'on nommera LoadJsGeneric.

Nous avons donc une organisation des classes qui ressemble à ceci :



[9] HyperText Transfert Protocol

Grâce à ce schéma, nous pouvons créer notre définition de liste, qui utilise l'objet Grid contenu dans sa classe parente, qui pourra implémenter une fonction pour être appelée par une requête HTTP, puisqu'elle hérite indirectement de la classe Action, et qui pourra être transformée en fichier JavaScript via sa classe parente directe.

L'architecture mise en place, il ne restait plus qu'à définir une adresse dans le fichier de configuration de Struts pour appeler notre classe à partir du menu. Pour se faire, il suffit de rajouter une ligne dans le fichier struts-config.xml :

```
<action path="/loadListJs" type="com.d2e.p2w.webapp.list.LoadJs">
    <forward name="success" path="/pages/genericList/list.jsp"/>
</action>
```

Après quoi il nous sera possible d'appeler notre fichier via l'url /loadListJs.do, et si tout se déroule bien, alors nous serons redirigés vers une page JSP qui contiendra le code JavaScript de notre liste.

Ci-dessous, un exemple de configuration d'une liste simple :

```
/**
 * Liste des ilots de parcelles
 */
public void createListIlotsParcelles() {
      log.info(LoginGeneric.getPrefixLog()
            + "Entering createListIlotsParcelles");

      Grid list = new Grid("listIlotsParcelles");
      list.setTitle("menu.ilotdeparcelles");
      list.addColumn("ilot.id", Type.INT).setWidth(150).setPrimary();
      list.addColumn("dossier.nom", Type.STRING)
            .setWidth(350)
            .setHeader("employee.dossier")
            .withCboFilter("Dossier")
            .withGroupBy();
      list.addColumn("ilot.nom", Type.STRING).setWidth(350).withFilter();
      list.addColumn("ilot.actif", Type.BOOLEAN)
            .setWidth(100)
            .withCboFilterById("YesNo")
            .withGroupBy()
            .setHeader("pcu_actif");
      list.Commit();

      // Requete Hql
      setHqlRequest("FROM ParRegroup ilot "
            + "LEFT JOIN ilot.paramDossier dossier");
}
```

Nous pouvons voir que pour chaque colonne, on va spécifier en suivant toutes les propriétés qui lui correspondent, ce qui nous permet de gagner du temps lors de la création de nos listes, et de pouvoir rajouter des propriétés supplémentaires pour chaque colonne plus tard si nous en avions besoin.

Maintenant que nous avons défini les propriétés de notre composant, et que nous pouvons appeler cette définition par le biais d'un menu, il nous fallait interpréter l'objet Grid, et le transformer en fichier JavaScript.

Nous allons donc maintenant étudier le mécanisme de transformation de cet objet Java en un fichier JavaScript grâce aux outils à notre disposition : Le Java, les pages JSP et les Java Bean qui feront le lien dynamique entre les deux.

2.4.4 Processus de génération du code JavaScript

En nous référant à notre schéma d'organisation hiérarchique des classes expliqué plus haut, on sait que LoadJs hérite de LoadJsGeneric. Donc la requête qui a appelé LoadJs peut être redirigée vers sa la classe parente, et ainsi accomplir les traitements pour transformer l'objet Java en JavaScript et insérer ces éléments dynamique dans la requête avant de retourner une réponse.

Il fallait bien faire attention à l'organisation des fichiers pour ne pas avoir à appeler des objets à gauche à droite et ne pas comprendre le cheminement que devait faire la requête. Grâce à cette organisation, on clique sur un menu, et l'enchainement des fichiers est logique.

Maintenant nous allons voir comment transformer notre code Java en JavaScript. Tout d'abord, nous pouvons créer un fichier JSP qui nous servira à afficher le résultat JavaScript souhaité.

Nous avons donc créé le fichier List.jsp, qui représentera le squelette générique de toutes les listes de l'application.

A l'intérieur de ce fichier, on va commencer par créer notre composant ExtJS représentant une liste, en lui donnant toutes les propriétés communes à toutes les listes.

Ainsi, nous aurons dans un fichier, la définition fixe du plus petit dénominateur commun du code JavaScript qui ne change pour aucune liste. Et c'est ensuite que nous ajouterons dynamiquement le code qui ne leur est pas commun, grâce à des outils intégrés aux pages JSP : les java bean.

Comment cela fonctionne :

- Une requête peut comporter plusieurs attributs. On peut d'ailleurs le voir dans beaucoup de sites web, par exemple si on cherche le mot « Supinfo » dans google, on va avoir une url qui ressemble à : `http://www.google.fr/search?q=supinfo`. Notre requête comportera alors un attribut qui se nommera « q » et qui aura pour valeur « supinfo ».

- Une action exécutée par Struts permet de faire passer une requête dans des classes Java, de faire des traitements, d'ajouter des attributs à cette requête, et de la renvoyer vers un fichier JSP.

- Un fichier JSP est compilé, et permet d'exécuter du code Java. De plus, on peut récupérer la valeur des attributs d'une requête grâce à des balises spécifiques appelées « bean ».

On va donc dans un premier temps créer des fonctions dans LoadJsGeneric pour transformer les informations de notre objet Grid sous forme d'une chaine de caractère représentant du code JavaScript.

Puis on va ajouter à la requête HTTP le bout de code JavaScript généré dans un attribut avec un nom correspondant. Par exemple, pour afficher le titre de notre liste, on va avoir une fonction pour récupérer le titre :

```
public String getTitreGrid() {
	return titreGrid;
}
```

Ensuite, on va faire passer notre titre sous forme d'attribut dans la requête :

```
request.setAttribute("titreGrid", UtilsGeneric.translate(getTitreGrid()));
```

Puis, enfin, nous allons l'afficher dans notre fichier JSP, afin que ce code apparaisse réellement à sa place dans du code javascript :

```
// Création du Grid
var grid = new Ext.grid.EditorGridPanel({
	…
	title : '<bean:write name="titreGrid"/>',
	…
});
```

Nous avons donc pu traiter la définition de la liste, la transformer en code JavaScript, la faire passer dans une requête HTTP, et l'afficher dans un fichier JSP afin de dessiner notre liste à l'écran de l'utilisateur.

Donc nous arrivons au stade où nous pouvons charger des listes, les afficher à l'écran, mais ces listes ne contiennent aucune données alors qu'une requête pour les charger a été spécifiée dans LoadJs lors de leur définition.

La raison qui nous a poussés à ne pas charger les données en même temps que la liste est que nous pouvons avoir des écrans avec beaucoup de listes. Par exemple, dans un écran décrivant les ordres de travaux de la journée dans un château de vin (l'écran le plus utilisé car il permet de répertorier toutes les actions effectuées dans la journée), nous avons une liste pour les employés, les matériels, les parcelles, les produits, les lots de vins, les mesures, les spécifications, et peut être d'autres à venir dans le futur.

Toutes ces listes sont construites lors du design de l'écran, afin que chaque composant prenne sa place dans les différents onglets, mais si on devait charger les données en même temps, alors le temps de chargement de l'écran le plus utilisé dans l'application augmenterait

considérablement, et les utilisateurs seraient las de passer leur temps à attendre que toutes les listes soient chargées alors qu'ils ne vont en consulter qu'une ou deux.

Il fallait donc lancer le chargement des données seulement quand on affiche la liste, et cela, ExtJS le permet grâce aux évènements JavaScript.

Une liste ExtJS permet de lire des données sous forme de tableau JavaScript, d'XML[10], ou de JSON[11]. Nous avons choisis d'utiliser le JSON, car il est aussi précis que du XML, mais est beaucoup plus compact et permet de faire transiter encore moins de données.

Nous allons donc maintenant étudier la méthode qui permet de faire passer la requête SQL pour charger les données, depuis l'endroit où on définit les propriétés de notre liste dans LoadJs, jusqu'à notre classe Java qui va transformer les données récupérées en JSON et les renvoyer à notre liste pour qu'elle les affiche.

2.4.5 La génération des données

Afin que nous puissions charger nos données au moment de l'affichage de la liste, il nous fallait tout d'abord trouver un moyen de stocker la requête SQL à exécuter pour les charger, et envoyer une nouvelle requête au moment de l'affichage de la liste pour recevoir les données.

Comme évoqué précédemment, nous pouvons déclencher des actions lors de certains évènements du client, par exemple lorsqu'il affiche la liste. Nous allons donc nous servir de cet évènement pour envoyer la requête de chargement des données vers une classe Java qui va

[10] Extensible Markup Language
[11] JavaScript Object Notation

s’occuper de retrouver la requête associée à la liste, de l’exécuter et de renvoyer les informations dans le bon format.

Nous avons donc créé la classe Java GenList pour gérer ces tâches.

Le premier point était donc de savoir comment transmettre la requête SQL entre les classes LoadJsGeneric et GenList par l’intermédiaire d’un fichier JavaScript List.jsp.

Après réflexion, deux méthodes semblaient possibles :

- Ecrire la requête SQL dans une variable JavaScript et la renvoyer plus tard vers GenList sous forme texte.

- Stocker les informations dont nous avons besoin pour charger les données en Session, et les récupérer plus tard grâce à un identifiant qui permet de reconnaitre la liste.

Etant donné que stocker la requête SQL dans une variable JavaScript (visible par le client si il affiche la source de la page) n’est pas du tout professionnel et comporte des failles de sécurité (SQL Injection, etc.), nous avons préféré le passage des données par la Session.

Pour bien comprendre le processus, voici comment fonctionne une session : C’est un fichier créé sur le serveur, propre à chaque client pour le site web. Ainsi, pour chaque utilisateur identifié sur le site, un fichier de session est créé où l’on peut stocker des informations. Une session a un temps de vie limité, si l’utilisateur n’effectue aucune action sur le site pendant un long moment, alors la session est invalidé, et l’utilisateur doit se ré-identifier.

Grâce à ce processus, nous pouvions alors stocker notre requête SQL en session dans notre fichier LoadJsGeneric en lui donnant un nom qui comporte l’identifiant unique de notre liste afin de pouvoir récupérer les informations plus tard dans GenList.

Voici le code pour stocker la requête :

```
session.setAttribute("request"+getUniqueId(), getHqlRequest());
```

Puis, dans notre page JSP, ExtJS permet de charger automatiquement les données lors de l'affichage de la liste, en donnant seulement quelques paramètres comme information, par exemple l'adresse vers laquelle envoyer la requête, et les informations qu'on veut lui passer en attributs.

Nous pouvons donc spécifier d'envoyer une requête vers GenList, avec l'identifiant unique de la liste, et ainsi nous pouvons retrouver nos informations en session grâce à un attribut fixe envoyé dans la requête pour charger les données.

Voici à quoi ressemble le code pour récupérer la requête SQL :

```
String query = (String)request.getSession()
      .getAttribute(request.getParameter("request"));
```

Une fois la requête récupérée, nous pouvions l'exécuter via Hibernate qui nous retourne les données sous forme d'objet, puis transformer facilement l'objet en JSON en associant chaque nom de champ à la valeur qu'il contient.

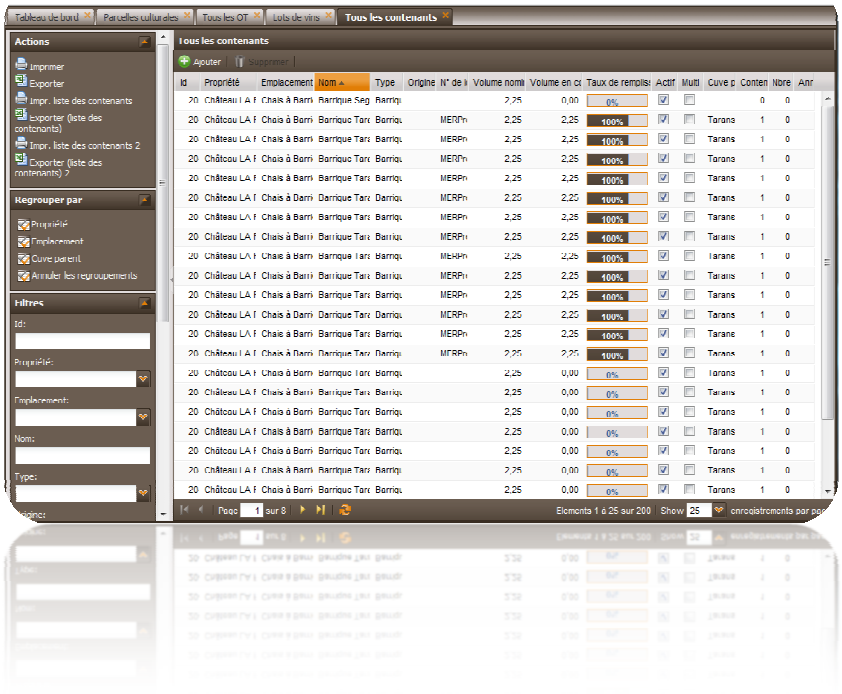
Puis, il ne nous restait plus qu'à écrire les données dans la réponse de la requête afin que la liste soit chargée et fonctionnelle.

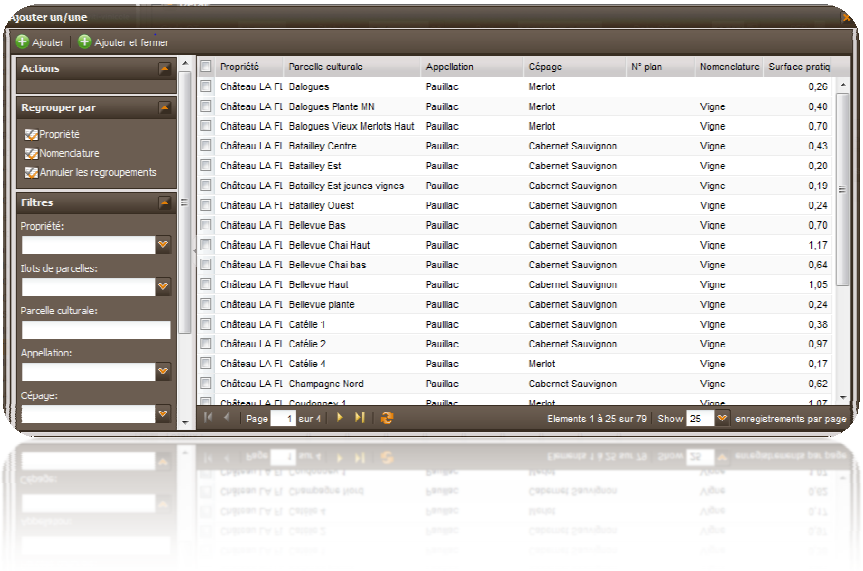
Notre générateur de listes étant maintenant terminé, nous pouvions migrer facilement les écrans du programme Access, en ayant juste besoin de la requête SQL anciennement effectuée, qu'on aura traduit au passage en utilisant les nouveaux champs de base de données (afin d'éviter de passer par les vues qui permettent aux champs qui ont changé de nom de toujours fonctionner), et des différentes colonnes que l'on veut afficher.

La totalité des listes de l'application a pu être migrée, de nouvelles ajoutées, et beaucoup d'améliorations sur l'ensemble des listes ont pu être réalisées facilement au cours du développement du logiciel.

Enfin, pour terminer, l'outil pour générer rapidement des listes était fonctionnel, mais les nouveaux développeurs ne sauraient pas comment l'utiliser. Il a donc fallu créer une documentation autre que la JavaDoc disponible via les commentaires dans le code, et une page a donc été créée sur le Wiki interne de la société afin d'expliquer la création d'une liste, et les différentes méthodes utilisables pour la paramétrer.

Ci-dessous, quelques écrans de listes de l'application.





Conclusion

Ce stage de fin d'étude m'a donc permis de mettre en application beaucoup de connaissances théoriques, mais avant tout d'utiliser beaucoup de méthodologie et de réflexion quant aux choix à faire.

J'ai eu la chance de travailler sur un projet complètement nouveau, et d'avoir la liberté de faire des recherches et établir des bilans pour proposer mes solutions au directeur de production.

J'ai donc pu choisir certains outils à utiliser, faire des recherches sur d'autres outils proposés et ainsi être certain que la migration de l'application se déroulerait sans avoir besoin de revenir en arrière pour changer de technologie.

Cela m'a permis d'apprendre l'utilisation d'un Framework : ExtJS, et d'utiliser beaucoup d'outils de développement que je n'avais jusqu'alors pas découvert.

La réalisation d'un générateur automatique de listes en JavaScript pour notre application a été un projet qui m'a pris à cœur, car c'est un projet qui évoluera en même temps que le logiciel et sur lequel nous pourrons encore apporter des améliorations au fil du temps.

De plus, ce stage m'a permis de mieux comprendre le travail en équipe et l'utilité de suivre un bon modèle de développement afin que chacun puisse poursuivre le travail d'un autre.

Pour finir, j'ai pu pendant mon stage modifier un programme Access afin de le lier à une base de données générée à partir de l'ancien schéma. Etudier les outils adaptés au développement d'une application Web riche, et créer un générateur de listes dynamique afin de migrer rapidement une application lourde vers une version web.

Remerciements

Je tiens à remercier particulièrement Emmanuel Thibierge, tout d'abord pour m'avoir proposé ce stage vraiment intéressant d'un point de vu professionnel, de m'avoir confié autant de responsabilités et de m'avoir permis de travailler dans d'excellentes conditions.

Je le remercie également d'avoir pris le temps de partager certaines de ses expériences et de nous avoir permis de visiter un château pour mieux comprendre le métier pour lequel nous travaillons.

Je remercie également toute l'équipe de D2E avec laquelle j'ai évoluée pendant mon stage, pour la bonne entende et avec qui travailler en équipe est un plaisir : Shengjie Tang, Maxime Riss, Alexandre Mokrane, Quentin Flayac, Sandrine Baloup, Jonathan Boucau, Louis Fleury, Virginie Rolland et Frédéric Camu.

Et enfin, je tiens à remercier Stéphanie Duboscq qui m'a soutenu pendant la rédaction de ce mémoire, qui m'a motivé, qui a pris le temps de me relire et de me corriger.

Annexe 1 - Documentation

Voici la documentation écrite dans le wiki sur l'utilisation de la classe Grid représentant notre composant de Liste en Java :

- addAlias(String aliasName, String sqlField, Type type) : ajout d'un alias pour une colonne lorsque le nom est trop compliqué. Le nom de l'alias (premier paramètre) ne doit comporter de caractère spécial (point, underscore, espace, virgule etc...). Le mieux est de respecter la même convention de nommage que les champs hibernate. (par ex : chaLotId, cuveVolNom).
- addable() : Combiné avec withAddLine, cette propriété permet de spécifier sur une colonne que le champs peut être ajouté.
- addable(AtomChamp.addMethods method) : Combiné avec withAddLine, cette propriété permet de spécifier sur une colonne que le champs peut être ajouté. method Enum permettant de choisir le type du champs qui va permettre l'ajout (type colonne, combobox, date).
- addable(String comboTable, String comboIdField, String comboValueField, String orderBy) : Combiné avec withAddLine, cette propriété permet de spécifier sur une colonne que le champs peut être ajouté. Le champs sera ajouté sous forme de combobox.
  - @param comboTable Nom HQL de la table ou chercher les données
  - @param comboIdField Nom HQL du champs de donnée
  - @param comboValueField Nom HQL du champs pour le texte visible
  - @param orderBy Nom HQL du champs qui servira à ordonner les données
- addable(String comboTable, String comboIdField, String comboValueField, String orderBy, String whereClause) : Combiné avec withAddLine, cette propriété permet de spécifier sur une colonne que le champs peut être ajouté. Le champs sera ajouté sous forme de combobox
  - @param comboTable Nom HQL de la table ou chercher les données
  - @param comboIdField Nom HQL du champs de donnée
  - @param comboValueField Nom HQL du champs pour le texte visible

    - @param orderBy Nom HQL du champs qui servira à ordonner les données
    - @param whereClause Clause where en HQL qui permet de filtrer les données de la combobox
- addAction(HashMap<String, String> action) : ajoute une action. Utiliser gerNewAction pour cérer une action.
- addButton(String libelle, String nom_fonction_appelle) : ajoute un bouton, lié à une fonction.
- addColumn(String data, String tableName, String idField, String valueField) : Ajoute des données qui seront éditables sous forme de combobox dans la liste.
    - @param data Nom du champs de donnée pour la sélection
    - @param tableName Nom HQL de la table où chercher les données pour l'édition
    - @param idField Nom HQL du champs qui servira d'id pour l'édition
    - @param valueField Nom HQL du champs qui servira de texte pour l'édition
- addColumn(String data, String className, String idField, String valueField, String orderByField) : Ajoute des données qui seront éditables sous forme de combobox dans la liste.
    - @param data Nom du champs de donnée pour la sélection
    - @param tableName Nom HQL de la table où chercher les données pour l'édition
    - @param idField Nom HQL du champs qui servira d'id pour l'édition
    - @param valueField Nom HQL du champs qui servira de texte pour l'édition
    - @param orderByField Nom HQL du champs pour trier les données de la combobo
- addColumn(String data, String className, String idField, String valueField, String orderByField, String whereClause) : Ajoute des données qui seront éditables sous forme de combobox dans la liste
    - @param data Nom du champs de donnée pour la sélection
    - @param tableName Nom HQL de la table où chercher les données pour l'édition
    - @param idField Nom HQL du champs qui servira d'id pour l'édition

    - @param valueField Nom HQL du champs qui servira de texte pour l'édition
    - @param orderByField Nom HQL du champs pour trier les données de la combobox
    - @param whereClause Clause Where pour filtrer les données à sélectionner dans la combobox
- addColumn(String data, Type type) : Ajoute une colonne à la liste
    - @param data Nom Hql/Sql du champs
    - @param type Enum pour le type du champs
- addColumn(String data, Type type, Class classObject, String functionName) : Ajoute une colonne à la liste, remplie grâce à une fonction
    - @param data Nom qu'on veut donner au champs dans le json
    - @param type Type du champs pour les filtres et les renderers automatiques
    - @param classObject Classe dans laquelle se situe la fonction (ex: Prime.class)
    - @param functionName Nom de la fonction à appeller pour générer les données
- addEvent(String varName, String key, String function) : permet d'ajouter un évènement en dehors de la grid.
- addFunction(String nom, String code_fonction) : ajoute une fonction.
- addHiddenField(String champs) : ajout d'un champs caché, n'apparaît que dans le jeason.
- addMenu(String libelle, String nom_fonction, String code_fonction) : combine les deux fonctions précédentes.
- checkBoxSM() : Ajoute une colonne de sélection par checkBox, désactive le RowColumnSelectionModel.
- Commit() : valide les composants de la grille. A utiliser en dernier bien évidemment.
- displayColumnFilters(Boolean display) : False pour désactiver les filtres des colonnes.
- displayStatusBar(Boolean display) : False pour ne pas afficher la bar de status en bas de grille.
- hideActionPanel() : Cache tout le volet avec les actions à gauche de la liste.
- required() : Spécifie que le champs est requis et doit être rempli lors de l'ajout.
- setAlign(String position) : Change l'alignement des données dans la cellule (position = left/center/right).

- setExpandable() : Active une colonne en mode tampon. A utiliser avec setExpandMode(Mode.Tampon);
- setExpandMode(Mode.MODE) : détermine la méthode d'affichage de la liste (peut-être FIXE, AUTO, TAMPON)
  - FIXE : grid avec scrollbar en bas si les colonnes dépassent.
  - AUTO : grid ne dépassant pas le panel.
  - TAMPON : fixe, avec une colonne mise en autoExpandColumn, par la méthode setExpandable().
- setEditable() : à rajouter si la colonne est éditable : date, texte, int.
- setEditor(String type) : pour spécifier le type d'éditeur de la colonne (textarea, etc...).
- setFieldToAdd(String name, String value) : Permet d'envoyer des champs qui ne sont pas dans la liste lors de l'ajout de données.
- setHeader(String translateName) : Spécifie le nom de la ligne de traduction pour l'entête d'une colonne.
- setHidden() : permet de cacher une colonne pour pouvoir appliquer un filtre.
- setOrderBy(String champs, String ordre) : choix du order by, sur la colonne souhaitée.
- setPrimary() : déclaration du champs correspondant en tant que clé primaire.
- setPrimary(String nom_champs) : si le champs n'est pas affiché dans la liste, on peut directement déclarer la PK.
- setRenderer(String function) : Change l'apparence des données dans une cellule.
- setTitle(String titre) : ajout du titre à la liste (inutile dans le cas d'une sous-liste).
- setUniqueId(String id) : permet d'attribuer un Id manuellement à un composant ExtJs.
- setWidth(Integer nb) : largeur de la colonne, en pixel.
- withAddLine() : Ajoute une line sous les noms de colonnes permettant l'ajout de données dans la liste.
- withCboFilter(String cboname) : ajout d'un filtre comboBox, à déclarer dans loadCombo.java
- withCboFilter(String cboName, String emptyText) : Ajoute un filtre de type combobox, filtre par le texte affiché, avec libellé en première ligne pour annuler celui-ci.

- withCboFilter(String cboName, String emptyText, String extId) : Ajoute un filtre de type combobox, filtre par le texte affiché, avec libellé en première ligne pour annuler celui-ci. extId : Id extJs unique permettant de sélectionner le composant en javascript avec Ext.getCmp()
- withCboFilterById(String cboName) : Ajoute un filtre de type combobox, filtre par l'id caché.
- withCboFilterById(String cboName, String emptyText) : Ajoute un filtre de type combobox, filtre par l'id caché, avec libellé en première ligne pour annuler celui-ci.
- withCboFilterById(String cboName, String emptyText, String extId) : Ajoute un filtre de type combobox, filtre par l'id caché, avec libellé en première ligne pour annuler celui-ci. extId : Id extJs unique permettant de sélectionner le composant en javascript avec Ext.getCmp()
- withDateFilterAfter() : Filtre par date en cherchant les résultats avec une date supérieure.
- withDateFilterBefore() : Filtre par date en cherchant les résultats avec une date inférieure.
- withGroupBy() : Ajout du regroupement correspondant à la colonne.
- withFilter() : Ajout du filtre correspondant à la colonne
- withTooltip() : Ajoute une Tooltip sur la cellule reprennant son nom et ses données

Annexe 2 - Visite de château

Ci-dessous, quelques photos prises lors de la visite du château Larmande-Soutard à Saint-Emilion, dans le but de nous faire comprendre comment travaillent les clients pour qui nous créons le logiciel.