Rapport Final Systeme

Document	**Rapport Final Système**
Version	**1.2**
Commencé le	**5 décembre 2006**
Dernière modification	**7 décembre 2006**
Statut	**Finale**
Client	**Enseignants du M2P GI**
Équipe	**DEBROUX Lionel** **FORNARA Romain** **KHLOUFI Samira** **TOKMEN Ali Savas**

Table des matières

1. But et portée du document

Le rapport final système du projet ECOM a pour but:

- De préciser l'état d'avancement de l'application FullManga.
- De mettre en évidence les points qui ont été étudiés en détail.
- D'évaluer le travail effectué par l'équipe.
- D'évaluer le respect du planning initial.
- De détailler les points ayant posé des difficultés.

Ce document est destiné:

- À l'équipe pédagogique

2. Glossaire

Voici l'explication d'une partie des termes techniques utilisés dans ce document:

Bean	Un composant logiciel écrit en Java
HTML	Acronyme pour HyperText Markup Language, c'est une technologie utilisée pour décrire un contenu (potentiellement dynamique). C'est la technologie qu'utilise la plupart des sites web.
CSS	Acronyme pour Cascading Style Sheeys, décrit le style d'un document.
AJAX	Acronyme pour Asynchronous JavaScript and XML, sert à ajouter de l'interaction via un site distant à un site web.
XML	Acronyme pour eXtensible Markup Language, c'est un langage de balisage. C'est utilisé en particulier pour décrire des configuration ou des requêtes - réponses.
JSP	Acronyme pour Java Server Pages, c'est un mélange de HTTP et de Java pour pouvoir générer des pages dynamiques.
JSTL	Acronyme pour JSP Standard Tag Library, c'est une librairie pour pouvoir écrire des JSP contenant des tags similaires au tags HTML au lieu du code Java, ce qui rend le code plus lisible et indépendant de l'architecture sous-jascente.
IHM	Interaction Homme-Machine, la science qui s'intéresse à l'interaction entre les humains et les machines: interfaces, modalités, ...
Fetch	Mot Anglais pour « obtention ». On l'utilise souvent pour parler de la phase de récupération des données d'une base.
Eager	Mot Anglais pour « ambitieux », inverse de Lazy (« faignant »). On l'utilise pour parler d'un système d'évaluation où tous les arguments sont évalués avant le traitement (que l'on ait besoin dans le traitement ou pas).
servlet	Objet Java sur un serveur J2EE qui peut être accédé via une URL HTTP.
URL	Acronyme pour Universal Resource Locator, décrit une adresse de ressource sur le web.
HTTP	Acronyme pour HyperText Transfer Protocol, c'est le protocole standard utilisé pour accéder à des sites web
J2EE (ou Java EE)	Acronyme pour Java 2 Enterprise Edition, c'est la plateforme utilisée par le serveur du site
SVN	Système de versionnage.

3. Stratégie de développement

Durant les étapes de conception et développement, nous avons suivi la stratégie suivante:

- Avant tout, l'équipe a été réunie pour que chacun puisse exprimer ses connaissances par rapport aux technologies à utiliser ainsi que par rapport au type de site à développer.
- Puis, les buts du site, le contenu à prendre en compte et les fonctionnalités nécessaires ont été identifiées via un questionnaire (donc une analyse de marché).
- Une fois cette étape complétée, l'architecture (la base de données et une maquette très grossière des fonctions accessibles de l'extérieur) a été tracée.
- L'application serait formée de trois parties:
  - La partie « stockage », qui stocke les données et les met à jour (donc aussi vérifie qu'il y a une certaine cohérence). Ce sont nos beans.
  - La partie « interface », donc les boutons (ou autre type de commande) pour dialoguer avec le site.
  - La partie « calcul », donc comment ces boutons sont liés au stockage.
- Le but a donc été d'arriver à avoir, avant la semaine bloquée:
  - Un stockage simple (donc qui stocke, mais ne vérifie pas la cohérence - on pouvait par exemple ajouter dans son panier des quantités négatives de produits) et une interface simple (du HTML très basique, avec du texte pur et des liens - pas de tableaux, pas de CSS, surtout pas d'AJAX, du JSP ou du JSTL).
  - La partie « calcul » qui fonctionne sans aucun problème.
  - Les divers documents (cahier des charges, dossier de conception IHM et dossier de conception système) prêts.
- Pendant la semaine bloquée, l'équipe a été scindée en trois (avec le temps passé dans chaque partie entre parenthèses):
  - La partie « stockage », qui a fait tout le nécessaire pour pouvoir avoir un stockage plus intelligent, qui entre autres vérifie des pré et post-conditions. (40%)
  - La partie « interface », donc toute l'apparence du site. (40%)
  - La partie « documents », donc les divers évaluations. (20%)

Ce planning, prévu dès le début, a été respecté.

4. L'état des composants et des fonctionnalités

1. Beans entité

Dans le dossier de conception système, sept beans entité avaient été définis:

- **ProductBean**: représente un produit vendu dans le magasin.
- **AuthorBean**: représente un auteur. Un produit a un ou plusieurs auteurs.
- **EditorBean**: représente un éditeur. Un produit a un éditeur.
- **CartElementBean**: représente un élément dans le panier. Un élément panier est un produit avec une quantité.
- **CartBean**: représente un panier. Un panier contient autant d'éléments panier qu'il veut.
- **OrderBean**: représente une commande. Contient un ensemble d'éléments de panier achetés, une date d'achat, etc.
- **AccountBean**: représente un utilisateur. Un utilisateur a un login, un mot de passe et peut sauvegarder des paniers, voir l'historique de ses commandes, etc.

Durant l'implémentation, nous avons remarqué que cette organisation ne nous permettait pas d'avoir les informations sur les produits de façon satisfaisante. De plus, les mangas ayant généralement plusieurs dizaines de tomes pour un même titre, stocker le titre était une mauvaise idée. Un nouveau bean a donc été crée:

- **ProductSeriesBean**: représente une série de produits et contient un titre et une description. Les ProductBean vont donc pointer vers un ProductSeriesBean pour les différents tomes du même groupe de produits.

a. Aspects fonctionnels

Tous ces beans ont été implémentés de façon satisfaisante:

- Leurs éléments et méthodes permettent d'avoir la totalité des fonctionnalités décrites dans le cahier des charges.
- Leur définition est correcte: les liens entre objets sont des liens avec des clés étrangères, les contraintes (unicité, etc.) sont définies avec des annotations.
- Leur stockage dans la base de données se fait correctement: le référencement, les recherches (inclues dans les sous-éléments) ainsi que les mises à jour fonctionnent..

b. Aspects non-fonctionnels

Si on regarde les aspects de sécurité et de performance:

- Les fetch de type « eager » sont minimales, ce qui renforce les performances.
- Les getters de beans ne transforment pas leur contenu avant de rendre un résultat: les Set et Collection sont par exemple rendus en tant que Set et Collection (et pas par exemple transformés en un tableau).
- L'accès aux sous-élements de chaque bean (par exemple aux paniers sauvegerdés d'un utilisateur) doit d'abord passer par le parent (donc dans cet exemple l'utilisateur en question). Ceci renforce la sécurité.

c. Conclusion

Le travail du côté des beans entité est donc terminé et validé.

On notera tout de même qu'un bean qui avait été prédéfini dans l'archive ECOM initiale, **ProductStoreBean**, n'a pas pu être enlevé: il ne nous sert à rien, il n'est pas référencé; mais le serveur J2EE n'est pas content (celui-ci plante en disant qu'il ne trouve pas **ProductStoreBean**) quand on l'enlève

2. Beans session

Dans le dossier de conception système, trois beans session avaient été définis:

- **EcomCustomerBean**: représente un client « normal », qui peut visiter la magasin, mettre des articles dans son panier et les acheter; mais n'a pas accès à la sauvegarde de paniers ou à l'historique des commandes.
- **EcomUserBean**: représente un client qui a un compte sur le site. Il a donc aussi accès à la sauvegarde de paniers, à une historique des commandes, etc.
- **EcomAdminBean**: représente un administrateur (que l'on avait supposé être le propriétaire du site). L'administrateur peut ajouter, modifier ou enlever des produits, auteurs, éditeurs et utilisateurs. Il peut aussi consulter l'historique des ventes, les pages les plus populaires, etc.

a. Aspects fonctionnels

Tous ces beans ont été implémentés de façon satisfaisante:

- Leurs méthodes permettent d'avoir la totalité des fonctionnalités décrites dans le cahier des charges.
- Chaque méthode a été testée de nombreuses fois (surtout durant l'implémentation du client léger).

b. Aspects non-fonctionnels

Si on regarde les aspects de sécurité, de performance et de convivialité:

- Leurs méthodes ont des mécanismes pour valider les préconditions (sur les arguments comme sur l'état du système) et jettent des exceptions si les préconditions ne sont pas satisfaites.
- Les exceptions jetées sont relativement précises, donc le client (qui a rentré de mauvaiss paramètres) comme le développeur (qui doit décrypter un plantage) sont aidés.
- Leurs méthodes utilisent des transactions.
- Finalement, ces beans ont passé avec succès un certain nombre de tests, surtout des tests de montée en charge et en capacités:
    - Notre base inclut près de 300 produits, avec chacun beaucoup d'attributs donc des attributs longues comme des descriptions (dans lesquelles les utilisateurs peuvent faire des recherches combinées ou croisées) ainsi que des images. En outre, son rendu est « instantané ».
    - Nous avons testé (et réparé de nombreux problème) en appliquant des tests simples de montée en charge (via des servlets).

c. Conclusion

Le travail du côté des beans session est donc terminé et validé.

3. Servlet

Nous avions défini un (et un seul) servlet. Ce servlet aurait les responsabilités suivantes:

- L'identification et la reprise de la session d'un utilisateur.
- L'analyse et la validation de la requête HTTP reçue.
- Le traitement de la requête.
- Finalement, la mise en place de contenu à rendre et l'appel au composant de rendu (JSP).

a. Aspects fonctionnels

Le servlet est en grande partie terminée:

- Toutes les opérations client et utilisateur (parcourir le magasin, faire des recherches, organiser le panier, acheter le panier, s'enregistrer, s'identifier, accéder aux panier sauvegardés, accéder à l'historique des commandes, etc.) ont été implémentées, testées et validées.
- Les opérations administratives (ajouters / modifier des produits, suivre les statistiques de ventes, etc.) n'ont pas été implémentées.

b. Aspects non-fonctionnels

Si on regarde les aspects de sécurité, d'integrité et de convivialité:

- Chaque session est identifiée par son identifiant de session, fourni par le serveur J2EE et qui est suivi que l'utilisateur ait les cookies activés ou pas.
- L'authentification et le traitement « global » est fait en un seul point (un seul servlet), ce qui permet de renforcer la sécurité (voir le point suivant) et l'integrité (voir le point d'après).
- Les beans d'une même session utilisateur sont regroupés au sein d'un même objet pour chaque utilisateur (qui est donc identifié par son identifiant de session), et les références (vers le panier, l'instance utilisateur, etc) sont locaux à chaque regroupement. Donc, il est impossible qu'un utilisateur puisse aller voir dans la session d'un autre.
- Une fois que la session d'un utilisateur est identifiée, le servlet entre dans un bloc synchronisé par l'objet session. Donc, on ne traite qu'une seule requête d'un utilisateur donné à la fois (on peut bien sûr traiter plusieurs requêtes: il suffit que ces requêtes proviennent d'utilisateur différents!). Ceci évite les incohérences (visibles surtout sous des conditions de montée en charge).
- Chaque exception jetée par les couches en dessous sont attrapées et redirigées vers les messages d'erreur et de recommendation (écrits à la main, de façon claire et concise) correspondants. Donc, quand une erreur survient, l'utilisateur voit un beau message d'erreur et de recommendation au lieu d'un « stack trace » (qui fait peur aux utilisateurs « normaux » et donne des idées aux utilisateurs « malins » pour hacker le site - donc qui ne sert strictement à rien -).

c. Conclusion

Le servlet n'a donc pas toutes les fonctionnalités requises, en outre les fonctionnalités qu'il propose ont été largement étudiées, testées et donc validées.

4. JSP

Les JSP sont les composants de rendu de notre application. Nous avons deux genres de JSP:

- Le JSP primaire (**main.jsp**), qui transporte les données fournis par le JSP vers les variables de page, met en place le contenu général (dispositions des items - haut, gauche, milieu, etc.) puis apelle les JSP de rendu pur. C'est le seul JSP appelé par le servlet et le seul JSP qui contient du code Java.
- Les JSPs de rendu pur, inclus par le JSP primaire, qui rendent dans leur partie le contenu à rendre. Par exemple, nous avons le JSP « du haut », le JSP « lister les produits », le JSP « panier », etc. Ces JSPs sont écrits en JSTL.

a. Aspects fonctionnels

Les JSPs sont totalement terminés:

- Toutes les interfaces définies dans le cahier des charges ont été implementées, testées et validées.
- Les interfaces sont compatibles avec les navigateurs suivantes:
  - Internet Explorer 5 (Windows 2000 SP4), Internet Explorer 6 (Windows XP SP2) et Internet Explorer 7 (Windows XP SP2)
  - Mozilla Firefox 1.0, 1.5 et 2.0 (Linux, Windows 2000 SP4 et Windows XP SP2)
  - Opera 8 et 9 (Windows 2000 SP4)
  - Konqueror 3 (Linux)

  En effet, le rendu du site sur ces browseurs est identique, du moment que les mêmes polices sont présents sur le système.
- La charte d'IHM a été respectée (voir le dossier d'évaluation IHM).
- Les navigateurs des PDAs ainsi que les navigateurs pour personnes avuegles sont capables de visualiser le site: la mise en page utilise des tableaux (reformattée automatiquement par les browseurs mobiles) et tous les images ont des textes alternatifs décrivant le contenu.

b. Aspects non-fonctionnels

Si on regarde les aspects d'extensibilité et d'indépendance par rapport aux divers aspects:

- Pour ajouter une nouvelle fonctionnalité, il suffit de:
  - Faire le JSTL correspondant
  - Mettre une nouvelle ligne pour que main.jsp le prenne en compte
  - Si nécessaire est, mettre à jour le servlet

  Le travail « informaticien »est donc réduit.
- Pour changer l'apparence du site, il suffit de:
  - Changer le fichier CSS pour les couleurs,
  - Changer le fichier **main.jsp** pour les dispositions relatives des éléments.

  Ce travail est un travail pûr de graphiste.
- Pour ajouter une nouvelle langue, il suffit de:

    - Créer un nouveau fichier d'internationalisation (une liste de phrases) et traduire les phrases nécessaires.

  Ce travail est un travail pûr de traducteur.
- De plus, la majorité de l'interface étant codée en JSTL, la partie présentation est indépendante du format des beans sous-jascents, même voire du fait que le serveur soit un serveur Java.

Les JSPs et les sorties HTML ont donc été implémentés de façon extensible, reformattable et indépendant de la plateforme et de la langue.

c. Conclusion

Le travail du côté des JSPs est donc terminé et validé. De plus, l'interface est totalement indépendante du browseur, de la plateforme du serveur web et même de la langue à utiliser lors du rendu ainsi que des paramètres régionaux (vu que l'internationalisation JSTL inclut déjà, par exemple, la conversion des dates ou des devises).

5. Analyse du travail fourni

1. Temps consacré

Le projet a été développé en utilisant des techniques de versionnage (outil SVN), et les fichiers de journal nous donnent une idée très précise sur la quantité de temps consacrée aux diverses étapes, ainsi que sur le travail effectif obtenu à la fin de chaque séance.

Conception système: 20 heures passés en groupe, donc 80 hommes heures
Développement système: 35 heures passés en groupe, donc 140 hommes heures

On notera que la partie développement système inclut les tests, et la partie conception système inclut l'édition de documents.

2. Performance de l'équipe

Nous remarquons que la planning prévu au début a été respecté:

- Tous les documents ont été rendus à temps
- Le lundi 4 décembre, le servlet complet ainsi que des beans et une interface simples étaient prêts.
- Le jeudi 7 décembre, toute la partie code et les divers documents étaient prêts.

Notre performance a donc été bonne. Elle a d'autant été bonne que la division initiale du travail était bien définie et que l'outil SVN nous a permis une synchronisation du travail de chacun de façon instantanée et sans peine.

3. L'apport aux membres de l'équipe

Les membres de l'équipe ont progressé dans de divers domaines:

- Dans l'étude de marché: les questionnaires, l'analyse de la concurrence, etc. Nous pensons avoir été proches de nos clients potentiels.
- Dans la conception générale d'une architecture à base de base de données: nous avons défini une base qui a pu satisfaire tous les besoins de façon performante.
- Dans la rédaction de documents: nous avons rédigé tous les documents ensemble, et ils respectent tous une certaine convention prédéfinie.
- Dans la découverte d'une architecture: J2EE est une très grande architecture sur laquelle nous n'avions pratiquement aucune connaissance. Nous avons donc, dans moins d'un mois, appris à utiliser cette architecture et avons produit une application dessus.
- Dans le codage: il y a quand même eu du code produit dans de divers langages: Java, annotations, HTML, JavaScript, JSTL, ... ce qui a agrandi les connaissances de chacun. De plus, nous avons aussi maniés des niveaux d'abstraction très hautes (par exemples, les pages web sont indépendantes du browseur, de l'architecture sous-jascente mais aussi de la langue).
- Dans le travail d'équipe: une fois la base terminée, chacun a travaillé sur une tâche dédiée et a mis en commun (via SVN) son travail. Nous avons, tous les quatre, travaillé de façon constructive, conviviale et efficace.

6. Difficultés rencontrées

Durant l'implémentation de cette application, nous avons rencontré diverses difficultés avec le plateforme de développement:

- **Configuration du serveur J2EE difficile**: Chaque composant du serveur J2EE est configuré via son fichier XML spécifique... Le problème est qu'il y en a beaucoup !!! En cas d'erreur, il nous a été très difficile d'en trouver la source. Les erreurs peuvent même avoir des sources « bizarres », comme par exemple un variable d'environnement « classpath » que JOnAS n'aime pas ou des paramètres régionaux qui font que JOnAS a du mal à retrouver des fonctions (comme les toUpperCase et toLowerCase ont été écrits sans spécifier de langue et que les divers paramètres régionaux peuvent capitaliser les lettres de façons différentes).
- **Annotations EJB non-standardisées de façon satisfaisante**: Il est difficile de savoir si les annotations EJB 3 doivent être mises sur les getters / setters ou sur les attributs. La diversité des solutions proposées sur les divers tutoriaux et forums indique que c'est probablement très dépendant de l'implémentation (la sous-version de la version du sous-composant du serveur J2EE).
- **Technologie JSP / JSTL non-standardisée de façon satisfaisante**: Il y a eu de grands changements dans JSP / JSTL à chaque sous-version, de plus l'activation et la configuration des composants reste très dépendantes de l'implémentation. Par exemple, nous avons dû faire un mélange de méthodes pour les anciennes et nouvelles versions de JSP pour pouvoir avoir le support JSTL (alors que l'on est supposé avoir la dernière version, la version de Tomcat utilisée étant récente).
- **Documentation J2EE très incomplète ou invalide**: Sun ne propose aucun exemple de code dans leur manuel (à comparer surtout avec MSDN où chaque méthode contient au moins un lien vers un exemple). De plus, les EJB et JSP ayant subit des changements majeurs à chaque version, une grande partie des exemples trouvées sur le web (inclu le site de Sun) sont en effet invalides.
- **Technologies J2EE très mal conçues**: Par exemple, l'internationalisation (donc une procédure où des caractères internationaux sont utilisés) en JSTL utilise des fichiers .properties (qui sont, par définition, en ASCII). Donc, on ne peut pas utiliser UNICODE dans ce fichier. Il n'y a pas trop de problèmes avec la langue française (si on suppose que la machine sur laquelle va tourner le serveur J2EE et la machine où le fichier .properties utilisent les même paramètres régionaux -ce qui n'est pas toujours vrai!), mais par exemple pour écrire « Bonjour » en Japonais, nous devons écrire **\u3953\u3093\u306b\u3061\u306f** ... Pas très convivial surtout si on pense que le fichier .properties pour l'internationalisation est supposé être lu par des traducteurs (non-informaticiens).

Nous avons aussi eu des problèmes dû à des facteurs externes:

- **Alarme incendie**: Il y a eu, mardi, deux fausses alarmes incendie (exercices ???); ce qui a déconcentré notre équipe.
- **Problèmes réseau**: Il y a eu, mercredi, un entretien des routeurs du campus. En conséquence, le réseau VPN (donc l'accès à internet) n'a pas fonctionné jusqu'à midi. Nous avons en partie contourné le problème en utilisant les connexions réseau des salles PC.
- **Problèmes de salle**: À plusieurs reprises, certains membres de l'équipe ont dû être refoulés des salles « réservées »... Problème au niveau de la synchronisation des emplois du temps des salles du côté de l'administration?

7. Conclusion et remerciements

Nous pensons avoir travaillé de façon efficace et correcte. Nous sommes fiers de notre travail.

Nous tenons aussi à remercier nos professeurs qui ont été là non seulement pour nous présenter les projets mais aussi pour nous aider en cas de problème.