



Projet de 6ème semestre 1999
Laboratoire des Systèmes d'Exploitation (LSE)
Section Systèmes de Communication, EPF-Lausanne

TRANSFERT DE LA BASE DE DONNEES DE LA BIBLIOTHEQUE DU LSE VERS LE WEB

Etudiant :

JOHANSSON Sylvain

Responsable:

WIESMANN Matthias

TRANSFERT DE LA BASE DE DONNES DE LA
BIBLIOTHEQUE DU LSE VERS LE WEB

Sylvain Johansson, tél. 079-4473201, Sylvain.Johansson@epfl.ch

© Lausanne, le 10.6.1999

Table des matières.

- **Introduction**
 - Introduction
 - Donnée du projet
 - Qu'est-ce qu'un servlet ?
 - Cahier des charges
 - Démarche suivie
- **Base de données**
 - Analyse de la base de données existante
 - Quelle base de données utiliser ?
 - Conception de la nouvelle base de données
- **Conception du site**
 - Structure générale du site
- **Package `ch.epfl.lse.biblio.johansson.util`**
 - Classes de bases.
 - Couche 1
 - Couche 2
 - Commentaires sur le code.
- **Servlets**
 - Structure générale des servlets
- **Problèmes rencontrés**
- **Conclusion**

- **Annexes**

Introduction.

Introduction.

La bibliothèque du LSE est composée d'environ 300-400 livres. Ces livres ont été intégrés dans une base de données au fur et à mesure de leur achat, celle-ci est actuellement au format FileMaker Pro 2.1. Pourtant cette base de données n'a pas été utilisée à des fins de consultation, elle est restée confinée sur le Macintosh de la secrétaire.

Le projet consiste à transférer cette base de données vers le serveur web du labo, et de permettre ainsi la recherche de livres, les démarches d'emprunt et de retour ainsi que l'administration de la bibliothèque en ligne.

Donnée du projet.

Initialement (selon la donnée originale du projet), il s'agissait de s'adapter à la structure existante. À savoir d'étudier la faisabilité de différentes stratégies :

- Transférer la base de données vers FileMaker Pro 3.0 et scripter FileMaker Pro 3.0 à l'aide d'AppleScript pour gérer l'interaction avec le web en faisant tourner cela sur un serveur Macintosh.
- Transférer la base de données vers FileMaker Pro 4.0 et utiliser les options offertes par FileMaker Pro 4.0 pour gérer l'interaction avec le web en faisant tourner cela sur un serveur Macintosh.
- Étudier d'autres possibilités.

Cette donnée a été modifiée en début de projet (par impossibilité de disposer d'un serveur Macintosh).

La donnée définitive se présente ainsi : exporter la base de données vers un SGBD (système gestion de base de données) Java et gérer toute l'interaction avec le web à l'aide de servlets.

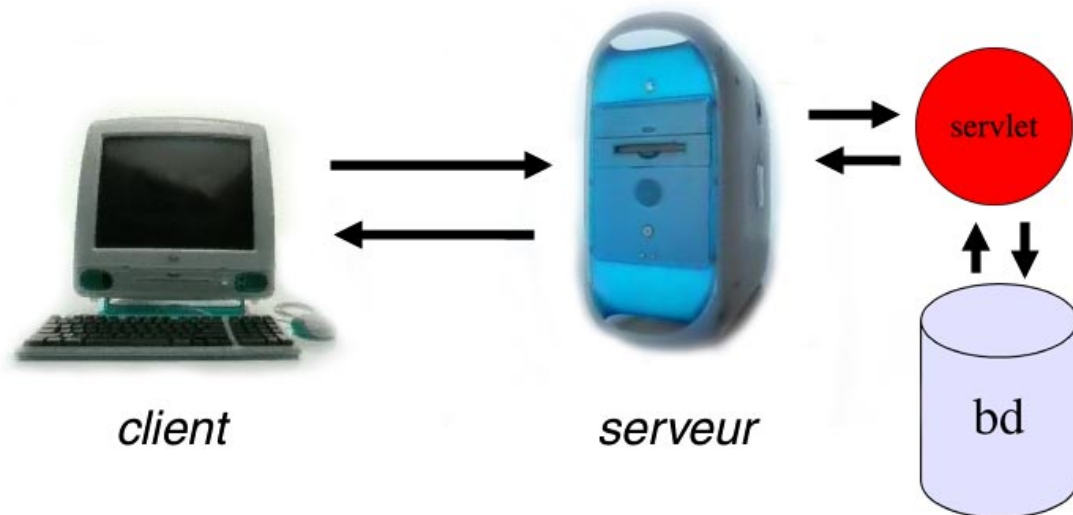


figure 1 : schématique du projet

Qu'est-ce qu'un servlet ?

Au premier abord un servlet peut être vu comme un "server-side applet". Il est chargé par le serveur dans sa machine virtuelle Java et exécuté (tout comme un applet est chargé par un navigateur puis exécuté). Le servlet est donc une sorte d'application Java. Le serveur qui reçoit une requête http pour le servlet la lui transmet. Le servlet exécute son code et retourne le résultat au serveur qui le transmet au client.

En somme, les servlets sont un excellent substitut de script CGI (voir la littérature sur les servlets pour une liste détaillée des avantages). Un des avantages qui n'est pas des moindres est la possibilité de pouvoir utiliser toute l'étendue du langage Java.

La programmation de servlets exige uniquement de prendre connaissance des principes de base, la lecture de quelques codes sources suffit largement à comprendre comment cela fonctionne. Pour qui a déjà programmé en Java, je dirai que c'est d'une grande accessibilité.

Concrètement pour ce qui est de la compilation d'un source de servlet, il suffit de disposer du package javax.servlet (disponible sur le site de Sun) et bien entendu d'un compilateur Java.

Pour pouvoir faire tourner des servlets, il faut simplement un serveur supportant ces derniers. Le Java Web server de Sun en est un exemple,

mais il existe également de nombreux modules d'extensions disponible pour la plupart des serveur existants (tel JRun pour Apache).

Ces quelques lignes se veulent simplement introductives, je conseille à quiconque est plus intéressé de consulter le web où de très nombreux articles permettent d'approfondir le sujet.

Cahier des charges.

Le cahier des charges a été établi en collaboration avec Matthias Wiesmann et se présente ainsi :

1. Éléments essentiels :

- Livres
 - Base de données structurée
 - Base de données peuplée selon état actuel de la bibliothèque
- Interface
 - Recherche / listes par :
 - auteur
 - titre
 - éditeur
 - mots clés
 - Modifications

2. Gestion emprunts :

- Gestion personnes
- Interface emprunt / retour

3. Optionnel :

- Exportation livre format *BibTeX*

4. Contraintes :

- Java 1.1
- Servlets

- SGBD: Instant DB
- Code portable
- Code solide
- Documenté (javadoc)

De ce cahier des charges se dégage la nette volonté d'obtenir une architecture en couche, ceci pour des évidentes raisons de souplesse future.

La gestion des personnes est voulue indépendante de celle des livres, afin de permettre une éventuelle fusion de données avec d'autres projets (par exemple le serveur bibliographique du LSE...).

Démarche suivie.

Ce projet a impliqué différentes activités préalables :

- La recherche de documentation sur des concepts théoriques inconnus de ma part.
- L'assimilation de ces différents concepts.

Voici la liste de ces concepts que j'ai eu à apprendre durant le projet :

Concept	Connaissances initiales	Connaissances acquises
Servlets	nulles	très bonnes
SQL	nulles	très bonnes
Programmation en Java	simples	bonnes
Programmation concurrente en Java	nulles	simples
JDBC	nulles	simples
Spécificités du protocole http	nulles	simples
JavaScript	nulles	basiques
Structure de base de données	basiques	simples
Format <i>BibTeX</i>	nulles	simples

Ensuite la programmation concrète :

- La création de la nouvelle base de données.
- La conception du site.

- Les classes Java nécessaires (ch.epfl.lse.biblio.johansson.util).
- Les servlets.

Base de données.

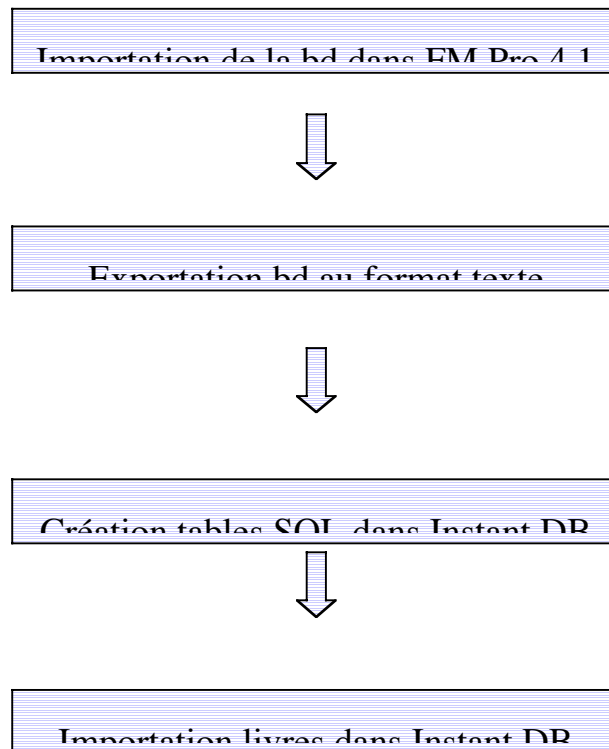


figure 2 : schéma création base de données

Analyse de la base de données existante.

La base de données originale contient les champs suivants :

- N° livre
- Titre
- Auteur(s)
- Éditeur
- Date d'édition
- Date d'entrée
- Mots clés

Ces champs ont été intégralement repris, avec leur syntaxe propre, c'est-à-dire celle adoptée lors du remplissage initial de la base. La reprise de cette syntaxe est source de quelques ennuis lors de l'exportation *BibTeX* par exemple (inversion des initiales et du nom des auteurs). Mais j'ai fait

le choix de conserver les données selon leur formatage initial et de traiter chaque éventuel problème au cas par cas dans le code.

La gestion d'emprunt et de retour nécessite des champs supplémentaires :

- Prénom emprunteur
- Nom emprunteur

Quelle base de données utiliser ?

Pour résumer en quelques lignes ce qui ne l'est pas, il existe un package `java.sql` (qui fait partie du JDK standard) qui permet d'interagir avec une base de données SQL depuis une application, un servlet ou un applet Java. Ce package contient JDBC (Java Database Connectivity), permettant ainsi des ordres SQL d'être envoyés à une base de données et les résultats récupérés depuis n'importe quel programme Java.

Ainsi tout SGBD proposant un driver JDBC est susceptible d'être utilisé depuis un programme Java. (Au passage, il faut noter le coup de génie de Sun. La majorité du marché utilisant ODBC (Open Database Connectivity) qui est la spécification de Microsoft pour l'accès aux bases de données. Il eut été illusoire d'espérer un développement massif de drivers JDBC de la part des différents fournisseurs de SGBD (Oracle, Sybase, Microsoft etc..) lorsque les investissements pour développer des drivers ODBC avaient déjà été faits. Sun a alors mis au point un pont JDBC-ODBC qui est en fait un driver JDBC utilisant des méthodes natives C (encore une des possibilités de Java...) permettant de faire appel à un driver ODBC pour accéder une base de données!)

Pour revenir à ce qui nous préoccupe, ceci signifie qu'un très grand nombre de SGBD peuvent ainsi être accédés depuis un servlet.

Dans le cadre de ce projet, sous les conseils de Matthias Wiesmann j'ai utilisé Instant DB, qui est un mini-SGBD écrit entièrement en Java, et surtout disponible gratuitement sur le web. Instant DB offre également un

driver JDBC compatible (idb). Instant DB implémente la plupart des fonctions de bases d'un SGBD, je n'ai en tout cas pas rencontré de limitations dans le cadre de ce projet.

Ainsi, à l'aide de JDBC, toute l'interaction avec la base de données est effectuée en faisant appel aux méthodes du package java.sql.

A noter que Instant DB est déjà utilisé par le LSE pour le serveur bibliographique.

Conception de la nouvelle base de données.

L'ensemble des champs nécessaires étant fixé, la conception du schéma entité-association de la base de données requise est possible, il est représenté en annexe [1].

J'ai ainsi pu déterminer les tables SQL nécessaires, soit :

livres TABLE

no_livre	auteurs	titre	editeur	annee_e dition	date_ent ree	mots_cle s	no_empr unteur
Unique Not null		not null					
int	varchar(1 00)	varchar(1 00)	varchar(5 0)	int	date	varchar(1 00)	int

figure 3 : livres TABLE

- L'attribut no_livre est l'identifiant simple et unique de la table livres.
- L'attribut no_emprunteur référence personnes.no_personne. Cet attribut est utilisé pour indiquer le statut d'un livre. Il est à zéro si le livre est disponible et au no_personne l'ayant emprunté le cas échéant.
- Les auteurs sont séparés par des virgules, les mots clés pas des tirets.

personnes TABLE

no_personne	prenom	Nom
-------------	--------	-----

Unique Not null	Not null	Not null
int	varchar(50)	varchar(50)

figure 4 : personnes TABLE

- L'attribut no_personne est l'identifiant simple et unique de la table personnes.
- Il est évident que l'attribut no_personne de la TABLE personnes est redondant, car l'attribut composé prenom+nom constitue un identifiant unique. Je l'ai pourtant rajouté pour des simples questions de commodité de manipulation de la TABLE.

Il est a noter que les deux TABLE sont indexées, livres sur no_livre et personnes sur no_personne.

Les bases de données et les TABLE ont été créés à l'aide d'un des utilitaires disponibles avec Instant DB (SQL Builder), les commandes SQL correspondantes sont représentées en annexe [2].

Conception du site.

Selon la donnée du projet, j'ai rapidement pu arrêter la forme et les fonctionnalités nécessitées par le site. Le plan du site est représenté en annexe [3].

Les principales rubriques étant :

- Recherche et emprunt d'un livre
- Retour d'un livre
- Administration
 - des livres
 - des personnes

Recherche et emprunt.

bibliothèque du
LSE

[accueil](#)
[recherche et emprunt](#)
[retour](#)
[administration \(restricted access\)](#)
[page principale du LSE](#)
[credits](#)

RECHERCHE

Recherche par critères.

titre :

auteur(s) : ex: Moss K., Schiper A.

éditeur :

année d'édition :

mots clés : ex: unix - solaris

résultats triés par : titre | auteurs | éditeur | année d'édition | mots clés |

livres concernés : tous | disponibles | empruntés

Recherche par listes.


tous les livres :	tous les livres disponibles :	tous les livres empruntés :
<ul style="list-style-type: none"> ● par titre ● par auteurs ● par éditeur ● par année d'édition 	<ul style="list-style-type: none"> ● par titre ● par auteurs ● par éditeur ● par année d'édition 	<ul style="list-style-type: none"> ● par titre ● par auteurs ● par éditeur ● par année d'édition

figure 5 : capture d'écran de l'index de recherche

La recherche démarre sur un index qui donne deux possibilités :

- Remplir un formulaire avec ses propres de critères de recherche, choisir l'ordre de tri (titre, auteurs, éditeur, date d'édition, mots clés) et choisir les livres concernés (tous, empruntés, disponibles).
Si plusieurs champs sont remplis, la recherche va porter sur tous ces champs (AND). Par contre si un champ est rempli avec plusieurs valeurs, la recherche portera sur une de ces valeurs (OR).
- Lister certains livres (tous, empruntés, disponibles) selon l'ordre de tri choisi.

Le résultat d'une recherche affiche une liste des livres satisfaisant les critères de recherche. La disponibilité de chaque livre est également indiquée, donnant accès à l'emprunt du livre ou au nom de son emprunteur le cas échéant.

bibliothèque du


[accueil](#)

[recherche et emprunt](#)

[retour](#)

[administration \(restricted access\)](#)

[page principale du LSE](#)

[credits](#)

index : |A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|

- titre : Distributed Algorithms (LNCS 725)
- auteurs : Schiper A.
- éditeur : Springer-Verlag
- année d'édition : 1993
- date d'entrée : 1993-11-29
- mots clés : algorithmes distribués
- statut : emprunté ([par qui?](#))

[bibtex export](#)

- titre : Programmation concurrente
- auteurs : Schiper A.
- éditeur : PPR
- année d'édition : 1986
- date d'entrée : 1988-08-25
- mots clés : programmation concurrente - langage de programmation disponible ([emprunter ce livre](#))

[bibtex export](#)

figure 6 : capture d'écran d'un résultat de recherche

Retour.

bibliothèque du


[accueil](#)

[recherche et emprunt](#)

[retour](#)

[administration \(restricted access\)](#)

[page principale du LSE](#)

[credits](#)

RETOUR

Sélectionnez-vous dans la liste.

--- liste des personnes ---

Si vous n'êtes pas dans la liste, lisez les conditions à la page d'accueil.

figure 7 : capture d'écran de l'index de retour

Le retour est voulu le plus convivial et le plus rapide possible. L'index de retour donne donc une liste des personnes présentes dans la base de données. En sélectionnant son propre nom, l'utilisateur se voit présenter une liste des ouvrages qu'il a empruntés avec possibilités de les rendre.

Administration.



ADMINISTRATION

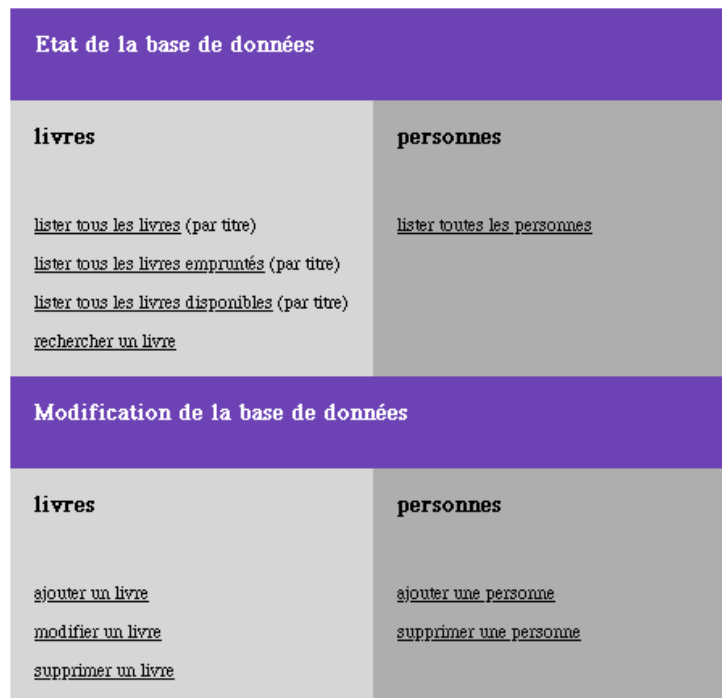


figure 8 : capture d'écran de l'index d'administration

Il s'agit d'un menu qui donne accès à l'ensemble des fonctionnalités de gestion de la base de données. Ajout, suppression et modification d'un livre, ajout et suppression d'une personne.

Ces opérations sont précédées d'un contrôle:

- La suppression d'un livre est impossible si celui-ci est emprunté.
- La suppression d'une personne est impossible tant que celle-ci a des livres empruntés.
- L'ajout d'une personne déjà présente est impossible.

Toutes les opérations impliquant une écriture dans la base de données (emprunt, retour, ajout, modification) passe impérativement au préalable par un écran de confirmation.

Intégrité des données rentrées dans les champs de formulaire.

J'ai choisi de ne pas vérifier l'intégrité des données par du JavaScript, il peut en effet très bien y avoir un utilisateur ayant désactivé cette fonctionnalité sur son logiciel de navigation.

Le contrôle est donc effectué dans les servlets mêmes, une donnée erronée suscitant le réaffichage de la page avec un message d'erreur.

Package ch.epfl.lse.biblio.johansson.util.

Le code a été développé selon un principe de couches. Le but étant de rendre chaque couche strictement indépendante des autres. Une modification interne à une des couches ne doit nécessiter aucun changement dans les autres.

Classes de bases.

Il s'agit premièrement de la classe Livres, une instance de cette classe permet de représenter un livre et fournit toutes les méthodes utiles à sa manipulation.

La deuxième classe de base est la classe Personnes. Une instance de cette classe représente une personne habilitée à emprunter un livre.

Lorsque le résultat d'une requête dans la base de données est un livre, celui-ci est placé dans une instance de la classe Livres, de même dans le cas d'une personne celle-ci est placée dans une instance de la classe Personnes.

1ère couche

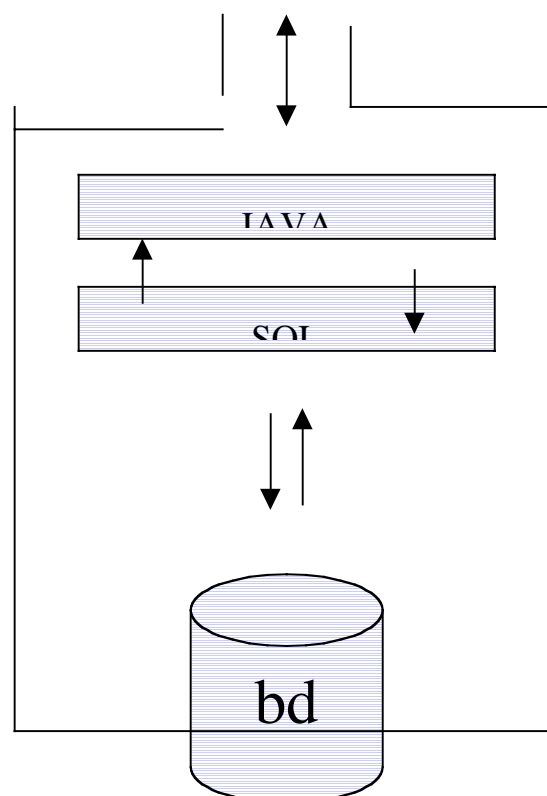


figure 9 : schéma de la 1ère couche

La première couche constitue le noyau du code. Elle comporte toute l'interaction avec la base de données, et de façon exclusive. Tout ce qui est formation d'une requête SQL selon certains critères, questionnement de la base de données, traitement des résultats (création d'un tableau d'instances de Livres par exemple) est encapsulé dans cette première couche. De telle sorte que l'interface visible de cette couche est constituée d'un nombre relativement restreint de méthodes Java permettant d'effectuer toutes les modifications ou tous les questionnements désirés en ne fournissant que les paramètres désirés. Par exemple la recherche de livres est effectuée en faisant appel à la méthode `researchInLivres()` qui prend en argument un objet Livres contenant les paramètres de recherche et retourne un tableau d'objets Livres représentant le résultat de la requête dans la base de données.

Les accès à la TABLE livres sont gérées par la classe `DatabaseUtilitiesLivres`, ceux à la TABLE personnes par la classe `DatabaseUtilitiesPersonnes`. Les méthodes générales ainsi que celles qui sont communes aux deux TABLE sont regroupées dans la classe `DatabaseUtilities`.

J'ai également développé une architecture lecteurs-rédacteurs pour les accès à la base de données, avec priorités égales aux deux. Plusieurs lecteurs sont autorisés simultanément (accès à la base de données en lecture) alors qu'un seul rédacteur est toléré (accès à la base de données en écriture). Cette architecture est regroupée dans la classe `DatabaseAccessRestriction`, elle est expliquée plus en détail en annexe [4].

L'interface visible de cette première couche est presque exclusivement des tableaux d'objets Livres ou Personnes.

2ème couche

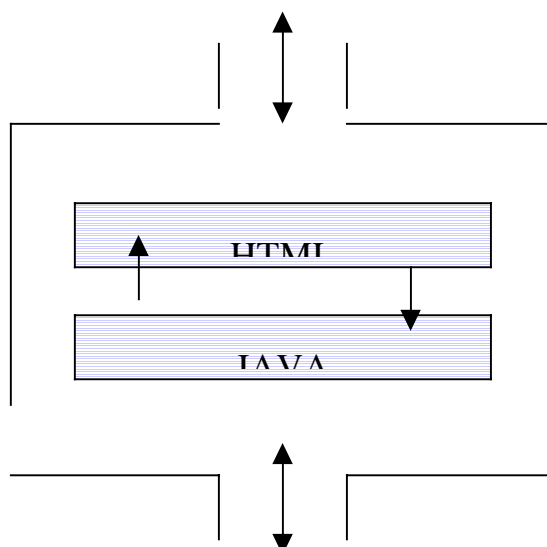


figure 10 : schéma de la 2ème couche

Cette couche fait le lien entre la première qui fournit le résultat d'une requête dans la base de données et le formatage des données en vue d'une publication Html.

Majoritairement cette couche fournit des instances de classe qui sont la plupart du temps des sous-classes de `lse.html.HtmlBasicObject`. Ceci dans le but de pouvoir aisément les inclure dans une page Html générée par la classe `lse.html.HtmlPage` à l'intérieur d'un servlet.

Ainsi on y trouve des listes de Livres, de personnes, des formulaires etc... La gestion du formatage en caractère Html est également comprise, voir les explications sur la classe `HtmlSyntaxedString` en annexe [5].

Cette couche contient également les outils pour faire le travail inverse. A savoir le décodage d'informations contenues dans des formulaires ou des liens Html, ceci au travers de diverses méthodes de traduction. Ainsi par exemple cette deuxième couche permet de traduire une requête Http d'un formulaire de recherche de livres en un objet de la classe `Livres` reflétant les paramètres contenus dans le formulaire.

Commentaires sur le code.

- Sécurité :

En plus de la satisfaction que procure un code relativement "propre", dans le cadre de servlets ceci se trouve être un impératif! En effet, il est très peu conseillé d'avoir un code instable lorsque celui-ci tourne sur la machine virtuelle du serveur web, ce qui est le cas avec des servlets.

J'ai donc pris un soin particulier à vérifier l'intégrité des données à chaque niveau. Ceci a principalement été fait par un traitement des exceptions, chaque zone sensible du code est entourée d'un "try {...} catch(Exception ex) {...}" afin de prévenir toute maladresse, et de permettre de au programme de supporter une erreur sans planter, dans la mesure du possible.

- Portabilité :

Les packages importés dans mon code sont ceux du JDK standard ainsi que certains du LSE. Aussi loin que je sache, aucun d'eux ne fait appel à une implémentation particulière qui pourrait limiter la portabilité du code. De plus, tous les tests ont été effectués sur au moins trois implémentation de la machine virtuelle Java différentes (MacOs Runtime for Java 2.1.2, Windows 98, SunOS 5.6) et je n'ai pas rencontré de problèmes.

- Lisibilité et documentation

Le code contient un maximum de commentaires, ce qui devrait permettre à un lecteur de s'y retrouver. Les commentaires d'en-tête ont été écrit de sorte à ce qu'ils soient compatible javadoc, ce qui permet d'avoir une javadoc complète des différentes classes et des servlets.

- Classe HtmlTitle

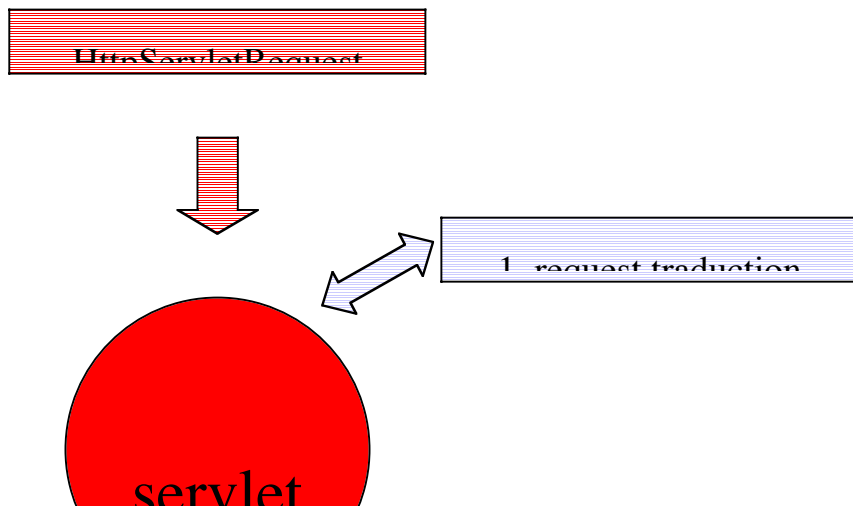
Cette classe représente les titres Html dans leur syntaxe standard, exemple :

```
<H3 align="center">.....</H3>
```

La classe HtmlTitle implémente la classe lse.html.HtmlObject et fournit ainsi des objets représentant des titres Html qui sont aisément intégrables dans une page Html générée par la classe lse.html.HtmlPage.

Cette classe aurait à mon avis tout à fait sa place dans le package lse.html du LSE.

Servlets.



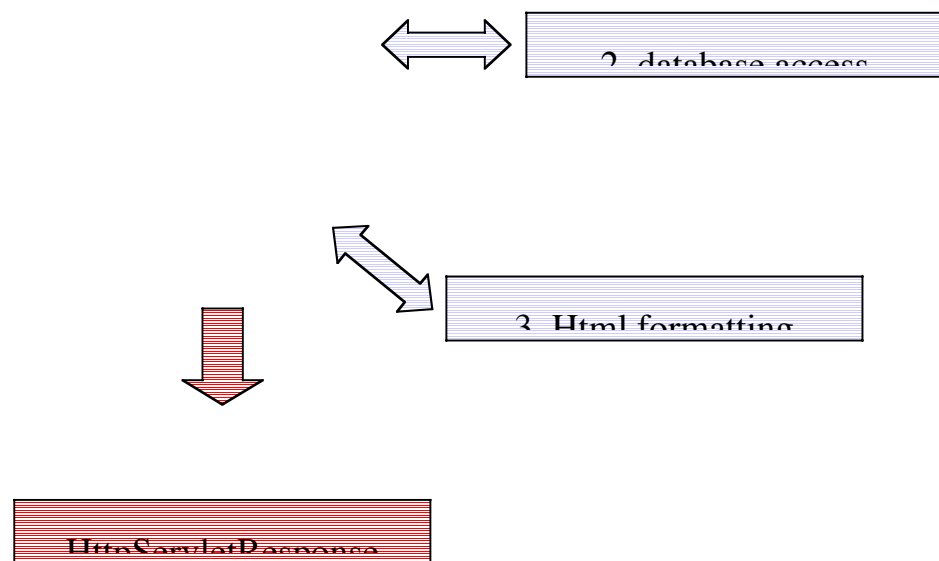


figure 11 : schéma des servlets

Les deux premières couches étant bien fournies, l'écriture des servlets s'est presque résumée à en faire simplement usage.

Le schéma général des servlets développés dans le cadre de ce projet est le suivant :

1. Les servlets font appel à la deuxième couche pour traduire les demandes de l'utilisateur en provenance du document Html source.
2. Une fois décodés, ces paramètres sont passés à la première couche qui s'occupe de sonder la base de données et de renvoyer une réponse satisfaisant la requête.
3. Cette réponse est ensuite repassée à la deuxième couche dans le but de la formater pour une publication Html. Les servlets génèrent ainsi une page Html dans laquelle est insérée le résultat de la requête.

Les servlets ne surchargent pas tous les deux méthodes doGet() et doPut(). Attention à ceux qui le font, parfois ce n'est pas du tout le même traitement qui est effectué.

Problèmes rencontrés.

Le premier problème est plus un inconvénient de développement lié aux servlets qu'un véritable problème. La gêne provient du test des servlets, autant les essais sur des classes peuvent être réalisés dans n'importe quelle machine virtuelle Java, autant les servlets exigent de faire appel à un serveur web (ou au `servletrunner` de Sun). L'ennui vient plutôt du fait que toute modification sur un servlet requiert de redémarrer le serveur (ou le `servletrunner`). Certaines autres solutions sont envisageables, mais il n'empêche que cela reste assez lourd.

La base de données originale fut source de quelques désagréments. Il m'a fallu corriger à la main (ou plutôt au clavier) des dizaines d'entrées. En effet, le remplissage s'est vraisemblablement fait sur plusieurs années et la convention syntaxique pour les auteurs n'était pas respectée tout du long...

J'ai également rencontré des problèmes de gestion des caractères accentués avec Instant DB. La visualisation du contenu de la base de données (à l'aide d'utilitaires fournis avec Instant DB) ou des requêtes imprimées sur le `stdout` portent à croire que les accents ne sont pas gérés correctement, alors qu'il n'en ait en fait rien.

D'autre part, pour ce qui de la syntaxe majuscule-minuscule, Instant DB ne gère malheureusement pas lui-même les différentes syntaxes (unix pour Unix par exemple). J'ai donc implémenté la première couche, de sorte à ce que les requêtes SQL soient effectuées sur plusieurs versions syntaxiques du même mot.

Il y a aussi certains développements inutiles de ma part. L'ensemble des fonctionnalités offertes par les classes standards de Sun étant relativement vaste, il m'est arrivé deux fois de mettre au point des méthodes faisant exactement le même travail que des méthodes déjà existantes!

Conclusion.

Au final, la partie visible du projet propose une interface simple et conviviale. L'ensemble des spécifications du cahier des charges a été pris en compte, et de nombreux tests ont été menés sur différentes machines virtuelles laissant supposer un fonctionnement correct.

Le service de consultation et d'emprunt de la bibliothèque du LSE sera ainsi accessible sur le web.

L'intégration avec d'autres projets du LSE est mise en évidence par le fait que la façon dont le projet a été structuré permettra une éventuelle future collaboration avec le serveur bibliographique. L'utilisation des classes des différents package lse.html est un autre exemple de complémentarité.

Ce projet m'a permis de connaître de nombreux sujets dans lesquels je n'avais pratiquement aucune connaissance, mais avant tout j'ai l'impression d'avoir été très bien conseillé et d'avoir appris à programmer bien mieux. La structuration en couche, la gestion des exceptions et autres sont autant de points ennuyeux à prendre en compte au début, mais j'ai réalisé à quel point ceux-ci deviennent indispensables lorsque la quantité de code est conséquente.

Annexes.

- [1] : schéma entité-association de la base de données.
- [2] : commandes SQL pour la création de la base de données.
- [3] : plan du site.
- [4] : explications sur la classe DatabaseAccessRestriction.
- [5] : explications sur la classe HtmlSyntaxedString.
- [6] : explications sur la classe LivresBibtexEntry.
- [7] : remarques diverses.
- [8] : environnement de travail.
- [9] : CD et contenu du CD.
- [10] : codes sources du package ch.epfl.lse.biblio.johansson.util.
- [11] : codes sources des servlets.