IUT Paris Descartes

2018-2019

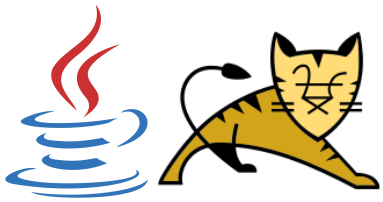
Applications Web Java

Projet Bibliothèque Web

Maud GELLEE — Groupe 201

Arsène LAPOSTOLET — Groupe 202

Table des matières



Page de Garde.....1

Table des matières.....2

Présentation de l'application.....3

Architecture du projet.....4

Concurrence.....6

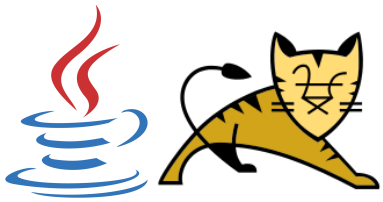
Solidité.....7

Diagramme de dépendances.....8

Amélioration possibles.....9

Bilan du projet.....10

I/ Présentation de l'application



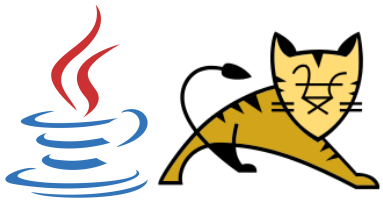
Introduction :

Lors de ce projet nous avons amélioré l'application réalisée pendant le cours d'Application Serveur Java en la transformant en une application Web à l'aide de Java Entreprise Edition. Nous avons transformé la bibliothèque en médiathèque (ajout de CDs et de DVDs) en abandonnant l'idée de réservation pour éviter toute difficulté supplémentaire, liées à la concurrence.

Deux types d'utilisateurs sont pris en compte, les bibliothécaires, qui peuvent ajouter des documents dans la médiathèque, et les utilisateurs qui peuvent emprunter et retourner des documents. Toutes interactions avec l'application se fait donc uniquement à partir d'un poste de la médiathèque.

Pour ce projet, des sources compilées nous étaient fournies avec le package *mediatheque*, qui servait d'intermédiaire entre le package persistance (code JDBC) et le package services (servlets), permettant ainsi de les rendre indépendant l'un de l'autre.

II/ Architecture du projet



Au cours de ce projet nous devions stocker les utilisateurs dans une base de donnée, afin de pouvoir gérer leurs connexions à l'application de façon persistante. Comme nous avions uniquement besoin de deux types d'utilisateurs, nous avons stockés les bibliothécaires et les clients au sein d'une même table *Utilisateur*, contenant les logins et les mots de passes nécessaires à la connexion ainsi qu'un booléen *isBibliothecaire* permettant de différencier les bibliothécaires des clients.

Pour la gestion des documents nous avons choisi de gérer l'héritage entre les documents et les spécialisations (DVDs, CDs, et Livres) en conservant la super-entité Document et les spécialisations. Avec cette méthode, la table *document* stocke le type de document auquel il correspond et les clefs primaires des spécialisations sont aussi des clefs étrangères liées aux clefs primaires de la table *document*. Ainsi, lorsque nous voulons emprunter et rendre un document il nous suffit d'accéder à la table Document, car cette dernière contient un booléen *emprunte* que nous modifions lors de l'emprunt ou d'un retour d'un document.

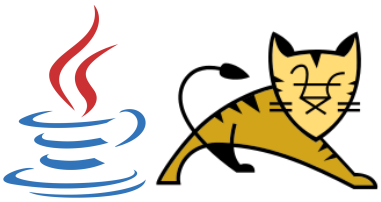
Les tables Utilisateurs et Document ne sont pas liées entre elles, car il n'est pas nécessaire pour ce projet (comme nous ne nous occupons pas de la réservation) de savoir quel Utilisateur à emprunter un Document.

Pour ce projet nous devions réaliser 3 package distinct et indépendants les uns des autres. Le package mediatheque qui nous était fournis, contient le domaine de l'application, le package services contient toutes les servlets et le package persistance contient le code JDBC et les classes correspondant aux informations stockées dans la base.

Pour plus de lisibilité et afin de stabiliser au maximum notre application nous avons séparés les packages services et persistance en plusieurs sous packages.

Nous avons séparé le package persistance en deux packages principaux. Le package persistance.bdd contient la classe nous permettant de nous connecter à la base de données, le point d'entrée du package avec la classe MediathequeData, ainsi que trois classes (DAO, Documents et Utilisateurs) qui contienne le code JDBC nécessaire à notre application. Le package persistance.modele contient lui toutes les classes utiles à l'application. Nous avons choisi de séparer les classes correspondants aux utilisateurs et les classes correspondants aux Documents dans deux packages distincts, persistance.modele.utilisateur et persistance.modele.document

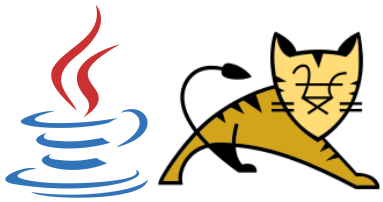
II/ Architecture du projet



(le package *persistance.modele.etatdoc* est présent car nous avons utilisé la pattern state pour la gestion des emprunts et des retours).

Nous avons séparé le package services en trois packages, le package *services.abonne* contient toutes les servlets nécessaire à l'utilisateurs abonné pour utiliser l'application, de la même manière, le package *services.bibli* contient toutes les servlets nécessaire au bibliothécaire. Enfin le package *services.auth* s'occupe de la connexion et de la déconnexion des utilisateurs.

III/ Concurrence

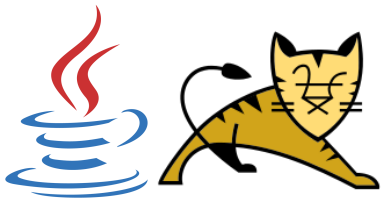


Lors de l'insertion de Documents, et à cause de notre modèle relationnel de données, nous faisons deux inserts consécutifs, un dans la table Document et un dans la table spécialisée concernée (DVDs, CDs, ou Livre). Il est donc nécessaire de gérer la concurrence lors de l'insertions de document. Pour ce faire nous avons créé une transaction, ce qui permet de ne pas mettre à jour la base tant que les données n'ont pas été insérées dans Document et dans la table spécialisée correspondante.

Une incohérence qui pourrait survenir serait par exemple lors de l'insertion de deux documents, les requête de l'insertion dans la table *document* pourraient être faites dans un ordre différente de ceux dans les sous table (en raison de l'accès concurrent) ce qui provoquerait une clé étrangère incohérente dans la table *document* à cause de lé génération de clé primaire dans les tables filles.

Pour la modification de l'état d'un document il n'y a pas de problème de concurrence car nous faisons un seul Update, c'est donc le SGBD qui gère la concurrence.

IV/ Solidité

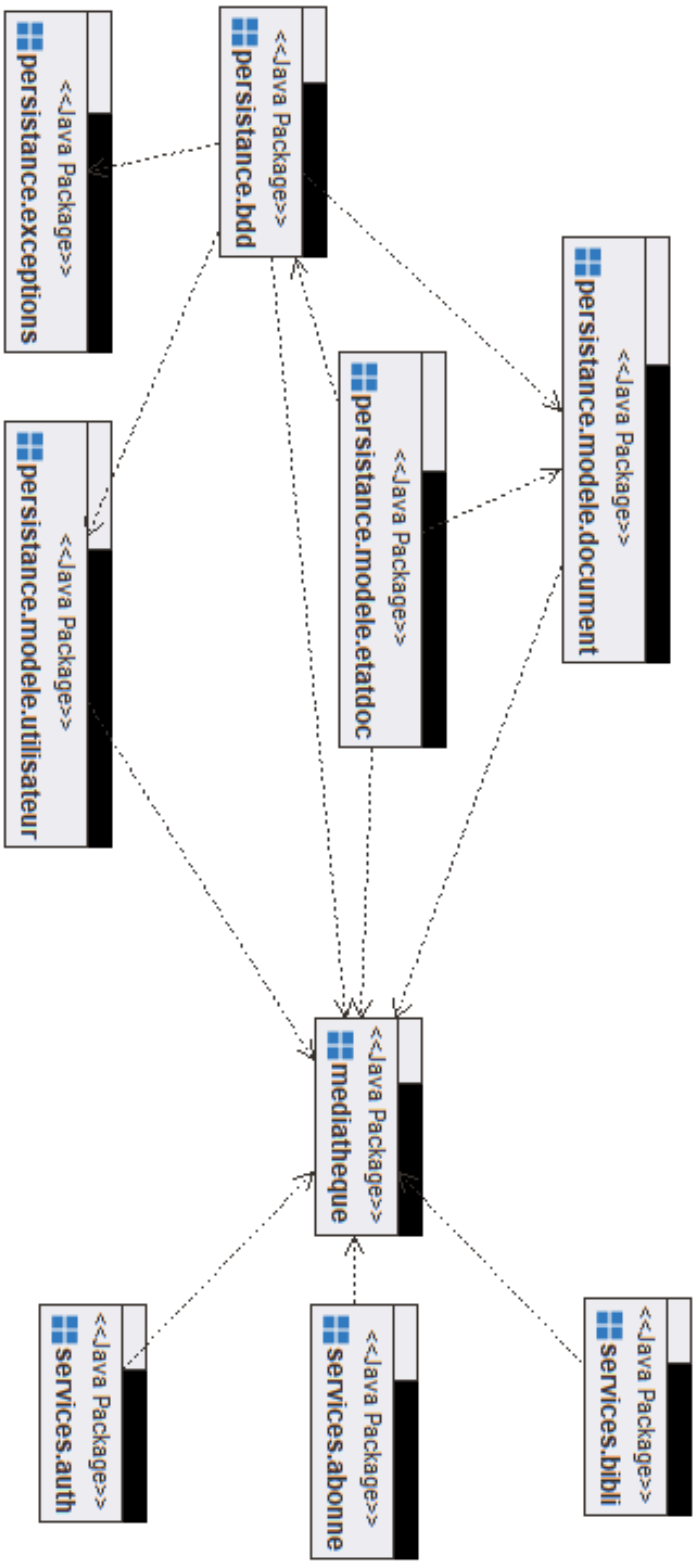


Pour gérer dans l'application l'emprunt et le retour de document nous avons utilisé le pattern State (délégation des services qui dépendent de l'état à une interface *EtatDocument*). Nous avons donc deux classes, Emprunte et Libre dans le package *EtatDoc*, qui s'occupent de gérer les changements d'état d'un document. Ainsi nous avons rendus le package document robuste à l'ajout d'état de document (comme par exemple un état réservé).

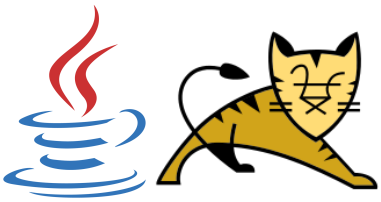
Dans une optique de polymorphisme, nous avons créé une classe abstraite *ADocument*, implémentant l'interface fournie Document, et les classes *DVD*, *CD* et *Livre* étendent *ADocument*. Ainsi nous avons pu factoriser du code et faciliter un ajout futur de type de document (comme l'ajout de BDs).

De la même manière nous avons créé une classe abstraite *AUtilisateur* implémentant l'interface fournie Utilisateur. Nous avons aussi créé une Factory *FactoryUtilisateur*, qui nous permet de déléguer la création de bibliothécaire et d'abonné. Nous passons à la factory les informations de l'utilisateur ainsi qu'un type, qui lui permet de définir s'il doit créer un abonné ou un bibliothécaire. Cela permettra de rajouter plus facilement des types d'utilisateurs.

VI/ Diagramme de dépendances



VII/ Améliorations possible



Améliorations faites :

Afin de sécuriser un peu mieux nos données, nous avons importé une librairie (bcrypt.jar) nous permettant de crypter les mots de passes des utilisateurs lors de leur stockage dans la base de données.

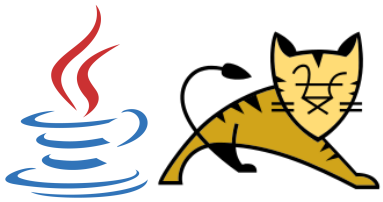
Enfin nous avons ajouté un service supplémentaire aux abonnés de la médiathèque, ils peuvent désormais, grâce à la servlet catalogue (package *services.abonne*) avoir la liste de tous les livres de la médiathèque ainsi que leur état actuel.

Pour l'aspect visuel de l'application, nous avons utilisé le framework CSS Material Design Lite afin d'obtenir un rendu visuel moderne, agréable et facile d'accès.

Améliorations possibles (cookies)

Afin d'améliorer l'ergonomie de notre application, nous pourrions utiliser des cookies. Ainsi nous pourrions stocker la variable utilisateur (que nous initialisons à la connexion d'un utilisateur dans la servlet *Login* de *services.auth*) non seulement dans la session qui se réinitialise donc à chaque fois que l'on quitte le site, mais dans des cookies, permettant ainsi à un utilisateur de ne pas se reconnecter à chaque fois. Cette amélioration serait une idée intéressante spécialement si on mettait en place le service de réservation, car il n'est pas logique d'implémenter ce genre de choses sur les ordinateurs de la médiathèque.

VIII/ Bilan du projet



Ce projet nous as permis de nous familiariser à la programmation Web en Java, et nous avons pu expérimenter l'utilisation des servlets et apprendre à utiliser JDBC.

Afin d'améliorer notre productivité et notre confort de travail collaboratif, nous avons utilisé le système de gestion de version Git. Nous avons donc utilisé l'hébergement gratuit framagit.org ainsi que le client graphique Fork afin de disposer d'une interface graphique.

Pendant ce projet nous avons pu apprendre une nouvelle utilisation de Java, mais aussi mettre en pratique nos précédents cours en faisant attention à la solidité et aux dépendance entre packages.

De plus, travailler en groupe reste une expérience très enrichissante, car nous avons la possibilité de nous encourager mutuellement. Nous sommes ainsi capables de surmonter plus facilement certaines difficultés et d'avancer plus rapidement en réfléchissant ensemble. De plus, en ayant déjà travaillé ensemble, nous étions très efficaces, connaissant nos points forts et nos points faibles mutuels.

Nous n'avons pas rencontré de difficulté particulière durant ce projet, nous avons cependant dû repenser notre utilisation des servlets. Nous utilisions la méthode *doGet()* afin de charger la JSP correspondante au service demandé, et avec la méthode *doPost()* nous traitions les données entrées par l'utilisateur. Pour les redirections d'une Servlet à l'autre, nous utilisions *request.getRequestDispatcher (…).forward(request,response)*. Cependant nous avons réalisés que cette méthode pouvait poser des problèmes dans certaines situations car ce n'était pas une réelle redirection. Nous avons donc modifié le code afin d'utiliser les servlets d'une meilleure façon, en utilisant la réponse cette fois ci la réponse des méthodes *doGet()* et *doPost()* grâce à la méthode *response.sendRedirect(…)*.