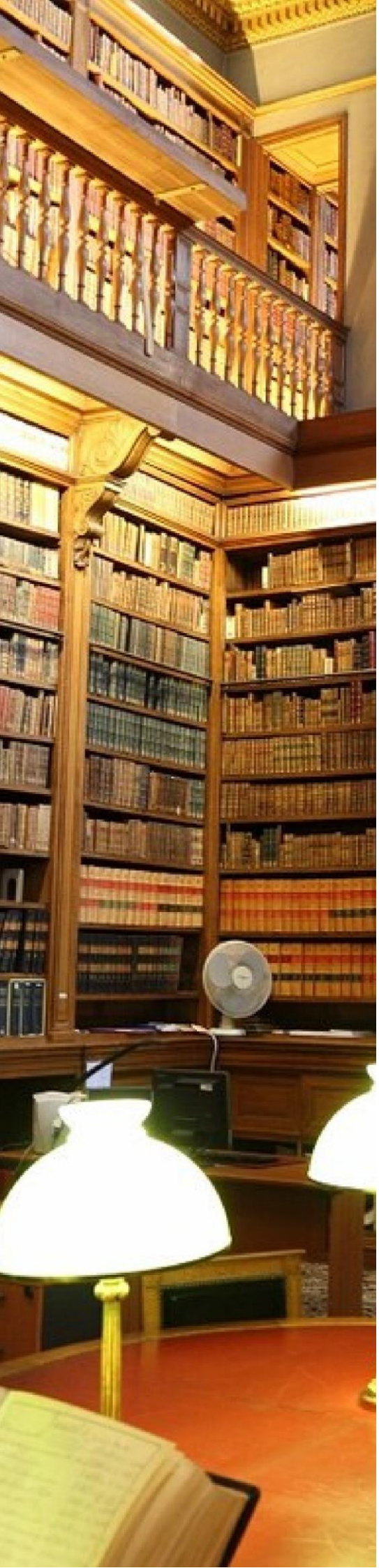


IUT PARIS
DESCARTES

RAPPORT
MEDIAWEB

Dylan RAKOTOARIVELO (201)
Julien ZHANG (201)

2020

Dylan RAKOTOARIVELO
Julien ZHANG

Table des matières

1. Présentation du projet

Le projet consistait à simuler certains services que peuvent proposer une bibliothèque. Plus précisément, il s'agissait de simuler les services d'emprunt et retour de documents (réservation n'est pas encore disponible sur cette version bêta).

Dans le cadre de la matière, il était important de mettre en relation le back-end, le front-end et la base de données (avec phpMyAdmin dans notre cas). Il était également important de mettre en œuvre un découplage strict entre les services servant aux échanges HTTP, la médiathèque contenant en grande partie le domaine stable, et enfin les éléments de persistance des données.

2. Structuration du code

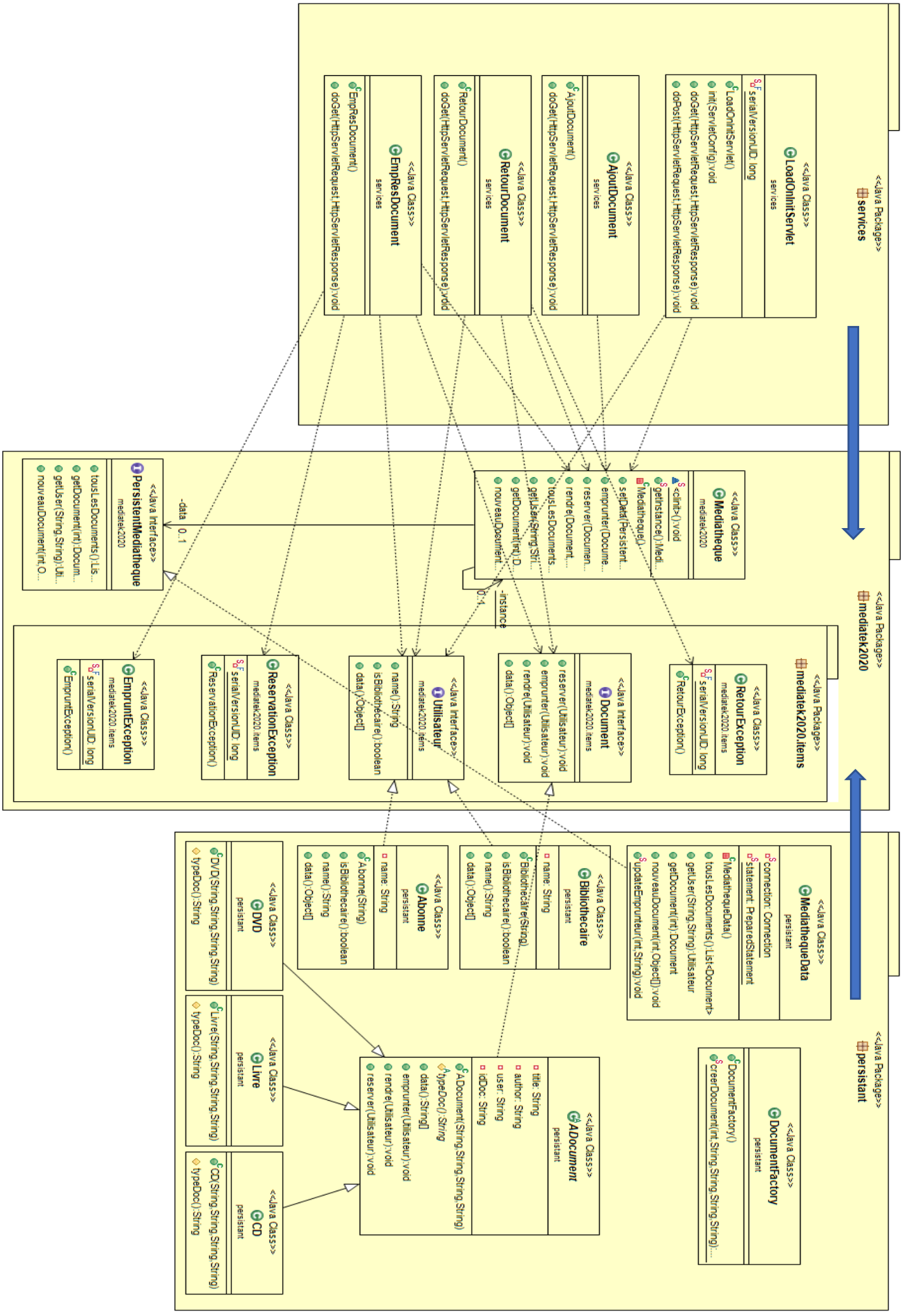
La classe Mediatheque doit interroger les données persistantes pour pouvoir faire le lien entre le package mediatek2020 et le package persistant. Aucun couplage ne doit subsister entre les packages services et persistance. De ce fait, la classe MediathequeData doit s'auto-déclarer à la classe PersistentMediatheque du package mediatek2020 par une injection de dépendances (en appelant la méthode setData dans son bloc static).

Notre architecture est alors composée de 3 packages structurés de la manière suivante :

services -> mediatek2020 <- persistant

avec mediatek2020 étant un package stable faisant le lien entre les servlets de services et les requêtes vers la base de données de *persistant* vers PhpMyAdmin. La mise en œuvre du découplage strict entre les 3 packages est bien respectée.

La page suivante vous montre, grâce au software **ObjectAid** la structuration du code en détail.



Dylan RAKOTOARIVELO
Julien ZHANG

3. Servlets, JSP, sessions et transformations objet-relationnel

Chaque classe du package services a son rôle. Il y a actuellement pour cette version bêta 4 classes (servlets) et chacune fait appel à la base de données via l'intermédiaire d'une méthode de la classe *MediathequeData.java* ou d'une méthode d'une autre classe du package *mediatek2020* puis vers une autre de la classe *MediathequeData.java.*

Lorsque Tomcat va être lancé au déploiement de la webapp, la servlet LoadOnInitServlet, placée dans le package services et dans le répertoire classes du WEB-INF, va être exécutée et initialisée. Pour montrer qu'elle a bien été initialisée, on effectue un affichage volontaire sur le terminal de Tomcat (une série de « * »).

Voici ci-dessous les fonctionnalités de chaque servlet :

- *LoadOnInitServlet.java*
  - **Initialiser la *MediathequeData***
  - **Authentifier**
  - **Créer la session**
  - **Maintenir la session**

- *AjoutDocument.java*
  - **Ajouter des documents**

- *EmpResDocument.java*
  - **Emprunter des documents**
  - **Réserver des documents**

- *RetourDocument.java*
  - **Rendre des documents**

Par ailleurs, chaque servlet et en plus de la classe *Document.java* du package *mediatek2020* sont utilisées dans les JSP qui ont pour rôle d'être l'IHM. Nous disposons de 5 vues JSP :

- **ident.jsp** :
  - *LoadOnInitServlet.java*

- **librarian.jsp** *:*
  - *AjoutDocument.java*
  - *MediatequeData.java*

- **user.jsp** *:*
  - *EmpResDocument.java*
  - *MediathequeData.java*

- **dashboard.jsp** *:* en fonction du statut de l'utilisateur, incluera **librarian.jsp** s'il s'agit d'un(e) bibliothécaire ou **user.jsp** s'il s'agit d'un(e) abonné(e).

- **header.jsp** : en-tête de la page qui l'incluera

4. Gestion de la base de données

Pour accéder à la base de données PhpMyAdmin, il est obligatoire de faire appel à la classe *MediathequeData.java* car elle est la seule à pouvoir recueillir, ajouter, modifier ou supprimer des données. Pour cela, il a fallu établir une connexion entre celle-ci et notre base de données en important le driver ***mysql-connector-java-8.0.15.jar*** et en faisant appel à la fonction **static *DriverManager*.getConnection(...)** une seule fois lors de l'instanciation de la classe. Ainsi, pour chaque méthode est attribuée une ou plusieurs requêtes selon le contexte. Enfin, pour optimiser ces dernières, les fonctions utilisent toutes un **PreparedStatement** qui rendra les requêtes à la fois plus rapides mais aussi plus lisibles.

5. Concurrence

La concurrence est un aspect primordial lorsqu'il s'agit de gérer plusieurs utilisateurs simultanément. Nous avons donc recherché quelles étaient les ressources partagées. Les plus évidentes étaient les documents, car il pourrait y avoir une incohérence si 2 utilisateurs veulent emprunter un même document au même instant. Ainsi, nous avons rendu la classe **ADocument,** threadSafe en disposant des blocs synchronized, dans les méthodes de cette même classe.

6. Bilan du projet

- **Les fonctionnalités que nous avons réalisées** :
  - Authentification
  - Création de la session
  - Maintien de la session
  - Ajouter des documents
  - Emprunter des documents
  - Rendre des documents
  - Messages d'alerte pour la réservation des documents
  - Concurrence

- **Les fonctionnalités que nous n'avons pas réalisées** :
  - Fermeture correcte de la session
  - Réelle réservation des documents