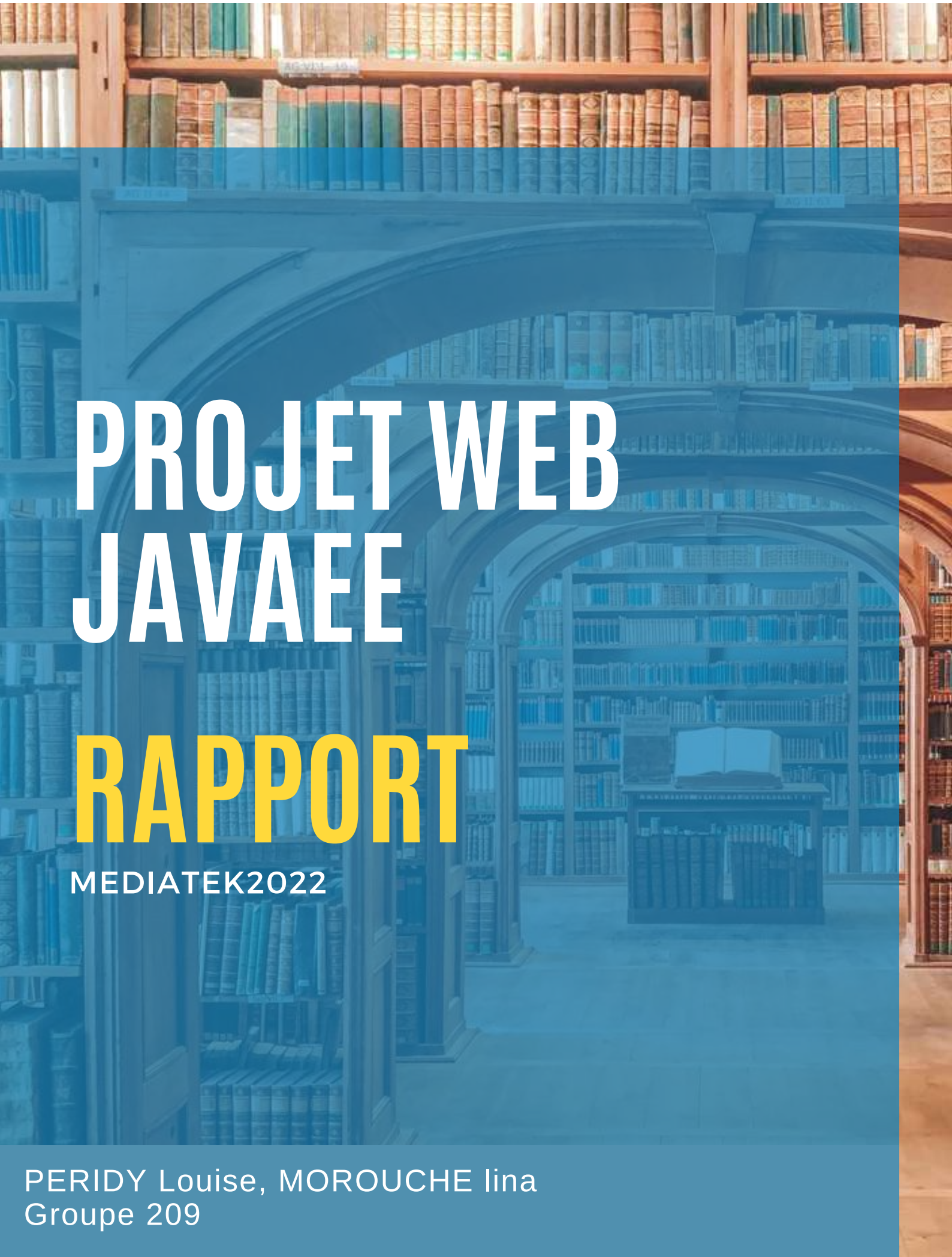
IUT de Paris - Rives de Seine
Université de Paris



PROJET WEB JAVAEE

RAPPORT

MEDIATEK2022

PERIDY Louise, MOROUCHE lina
Groupe 209

Sommaire

PERIDY LOUISE, MOROUCHE LINA
GROUPE 209

Get Started : installation de notre projet

Étapes pour installer notre application :

JAVA

- Créer un dossier projettomcat dans WebApp)
- Créer d'un projet Eclipse Java 1.8 dans TomCat
- Importer les 3 libs du Zip
- Copier les fichiers sur Eclipse

On a utilisé Uwamp pour se connecter à PhpMyAdmin, voici le code sql que l'on a mis dans une base qu'on a appelé jee_pj :

```
CREATE TABLE `document` (
`Id_Document` varchar(50) NOT NULL,
`Nom` varchar(50) NOT NULL,
`Id_Utilisateur` int(11) DEFAULT NULL,
`type_Doc` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `document` (`Id_Document`, `Nom`, `Id_Utilisateur`,
`type_Doc`) VALUES
('1', 'Les Ronces', NULL, 'Livre'),
('2', 'Noir Volcan', NULL, 'Livre'),
('3', 'Stupeflip The Hypnoflip Invasion', 1, 'CD'),
('4', 'Submarine (2010)', 1, 'DVD');

CREATE TABLE `utilisateur` (
`Id_Utilisateur` int(11) NOT NULL,
`nom_Utilisateur` varchar(32) NOT NULL,
`mdp_utilisateur` varchar(50) NOT NULL,
`estBibliothecaire` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

GetStarted : installation de notre projet

```
INSERT INTO `utilisateur` (`Id_Utilisateur`, `nom_Utilisateur`,
`mdp_utilisateur`, `estBibliothecaire`) VALUES
(1, 'Louise', '1234', 0),
(2, 'Lina', '123', 1);

ALTER TABLE `document`
ADD PRIMARY KEY (`Id_Document`),
ADD KEY `Id_Utilisateur` (`Id_Utilisateur`);

ALTER TABLE `utilisateur`
ADD PRIMARY KEY (`Id_Utilisateur`);

ALTER TABLE `document`
ADD CONSTRAINT `document_ibfk_1` FOREIGN KEY (`Id_Utilisateur`)
REFERENCES `utilisateur` (`Id_Utilisateur`);
```

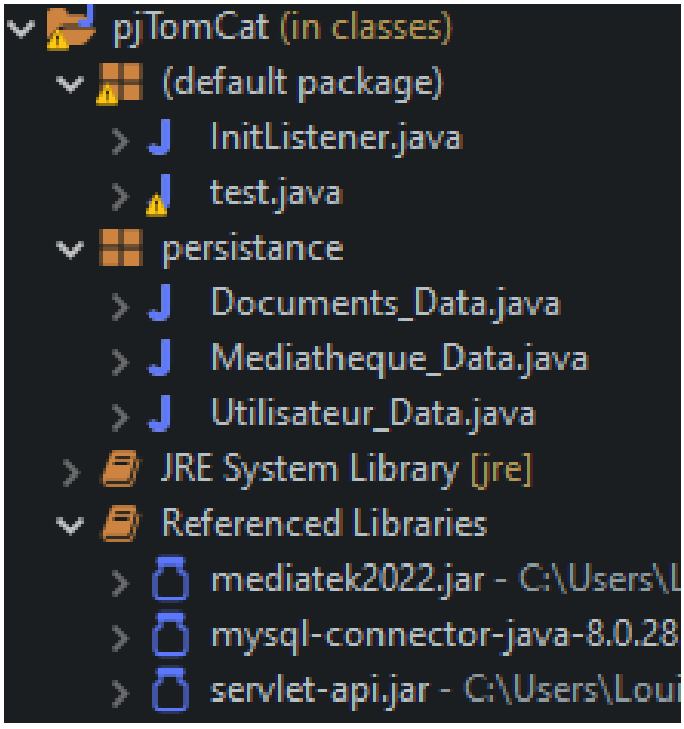
Introduction

Dans le cadre de notre cours de JavaEE, nous avons dû coder une application web permettant d'accédeer à un service de Médiathèque en ligne. Deux types d'utilisateurs peuvent s'en servir : les abonnés et les utilisateurs.

Pour cela, nous avons utilisé Java 1.8 pour coder les requetes à la base de données ainsi que la gestion des connexions. De plus, les JSP ont été utilisés pour afficher les pages web.

1. Structure de code et injection

Notre application se compose de JSP et de code java. Nous avons mis les JSP en dehors du dossier WEB-INF et à l'intérieur de ce dernier nous avons séparé les classes et les lib pour simplifier l'oganisation.



Au sein des lib, on a utilisé mediatek2022.jar, my_sqlConnector et servel.api et dans les classes on a mis les services de la persistance qui sont directement reliés au jar grace à la gestion des dépendances. On aura dans le code java le package de la persistance et celui des lib avec le jar de la mediatek. Le bloc static de Mediatheque_Data s'exécute au lancement de l'application grace au Listenner et permet de l'injection de dépendance dans Mediatek2022.

2. Utilisation de JSP et session

Nous avons utilisé des JSP pour afficher les différentes pages de la médiathèque : cinq en tout.

- Index.jsp : formulaire pour s'identifier.
Un utilisateur peut se connecter au service, si les identifiants ne sont pas corrects il sera redirigé vers un message d'erreur qui lui permettra de retenter.

- Authent.jsp : Page d'accueil.
Elle va permettre à un abonné de voir les libres disponibles directement, de cliquer sur un bouton pour voir ceux qu'il possède et/ou faire un retour ainsi qu'un bouton pour se déconnecter
La bibliothécaire va voir la liste de tous les livres ainsi qu'un bouton pour mener à un formulaire.

- Retour.jsp : Retourner un livre.
La page affiche les livres possédés par l'abonné ainsi qu'un bouton pour les rendre, libérant ainsi la clé étrangère de la table Document.

-Emprunt.jsp et retournerPage.jsp : actions.
Ces pages appellent les actions de retour et d'emprunt : elles n'affichent rien mais ramènent sur d'autres pages après avoir effectué les actions demandées.

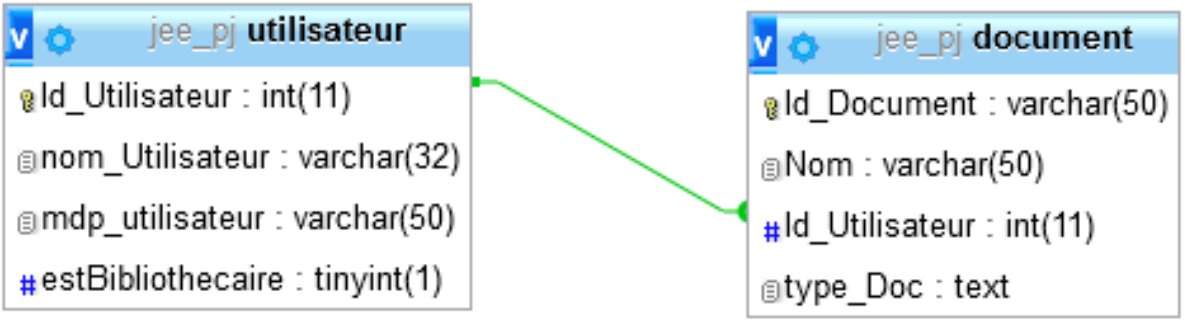
-Gestion.jsp : services de la bibliothécaire.
Cette page va etre dédiée à la bibliothécaire qui pourra ajouter un document dans la base de données à l'aide d'un formulaire dans lequel on pourra ajouter toutes les informations relatives aux documents.

Les sessions sont gérées à l'aide de l'objet session. A chaque connexion on va récupérer l'user avec Mediatheque.getInstance().getUser et le mettre dans l'objet session après avoir vérifié s'il n'existait pas déjà. Lorsqu'on cliquera sur le bouton déconnecter, on sera redirigé vers index.jsp.

Cette redirection appellera <%session.invalidate();%> ce qui permettra la fermeture de la session courante pour éventuellement en ouvrir une nouvelle.

3. Les objets et base de donnée

Nous avons utilisé PhPmyAdmin pour configurer notre base de donnée. Sur le site, nous avons configuré 2 classes : Utilisateur et Document.



On a créé 2 clés primaires Id pour les 2 classes afin de simplifier la façon de les répertorier et on a une clé étrangère Id_Utilisateur de la table utilisateur : avec cette disposition, on peut facilement voir si un document est disponible. En effet, si sa clé étrangère est nulle, alors il est disponible.

A l'origine, nous avions créé un attribut quantité pour un document mais finalement nous avons décidé de créer un document pour chaque livre. En effet, dans une vraie médiathèque on a plusieurs livres avec de états différents (neuf, abimé etc) et il nous semble plus réaliste de se pas tous les rassembler dans un même int.

Pour passer des objets entre les JSP et les java, nous avons décidé de créer des classes très similaires à la base de donnée. Ainsi, pour récupérer une liste de documents, il suffira d'appeler Mediatheque.getInstance().tousLesDocumentsDisponibles();

Utilisation de l'objet Data :
En particulier, on a utilisé la fonction Object[] data() astucieusement pour récupérer à la fois l'id de l'utilisateur avec la première case de l'objet (jusqu'ici on avait seulement le nom avec public String name()) mais surtout la liste de ses possessions. En effet, data[1] appelle la fonction livre Empruntes qui va récupérer tous les livres de l'utilisateur.

Jeu d'essai :
Pour tester chez nous, on a pris 2 types d'utilisateur : Bibliothécaire et Abonne ainsi que des documents disponibles et non disponibles.

	Id_Utilisateur	nom_Utilisateur	mdp_utilisateur	estBibliothecaire 1
Modifier Copier Effacer	1	Louise	1234	0
Modifier Copier Effacer	2	Lina	123	1

	Id_Document	Nom	Id_Utilisateur	type_Doc
Modifier Copier Effacer	1	Les Ronces	*NULL*	Livre
Modifier Copier Effacer	2	Noir Volcan	*NULL*	Livre
Modifier Copier Effacer	3	Stupeflip The Hypnoflip Invasion	1	CD
Modifier Copier Effacer	4	Submarine (2010)	1	DVD

Jeu d'essai Java :
La classe test dans le default package sert à tester au sein d'Eclipse le fonctionnement du code avant de l'implémenter et de l'afficher dans des jsp. Ainsi, on voit facilement les erreurs potentielles pour les corriger plus facilement.

```
Utilisateur user = Mediatheque.getInstance().getUser("Louise", "1234");

System.out.println(Mediatheque.getInstance().tousLesDocumentsDisponibles());
Mediatheque.getInstance().getDocument(1).emprunt(user);
System.out.println(Mediatheque.getInstance().tousLesDocumentsDisponibles());
Mediatheque.getInstance().getDocument(1).retour();;
System.out.println(Mediatheque.getInstance().tousLesDocumentsDisponibles());
```

```
<terminated> test [Java Application] C:\Users\Louise\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_14.0.2.v2020
trouvee
connexion effectuée
[<br><br>Document id :1<br>Nom : Les Ronces<br>Type : Livre<br><button onclick="v
[<br><br>Document id :2<br>Nom : Noir Volcan<br>Type : Livre<br><button onclick="
[<br><br>Document id :1<br>Nom : Les Ronces<br>Type : Livre<br><button onclick="v
```

4. Les points à améliorer

- La disposition des JSP : on pourrait réorganiser le code
- La gestion des requêtes : une pré-compilation aurait été plus sécurisée
- La gestion des multi-users : l'appli tourne sur un serveur et peut créer des soucis de concurrence (+ gestion des IP)