Projet MediaWeb - Stefan Radovanovic

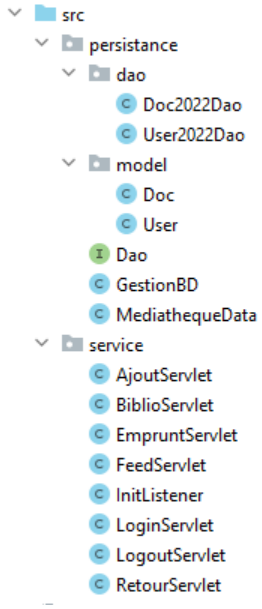
Rapport de projet Mediatheque JavaEE, programmation répartie

> ⚠️ Ce projet peut être testé **rapidement**, il suffit juste de déplacer la web-app dans votre tomcat (SGBD sur **serveur distant** et JDBC mariaDB dans le fichier **lib**) et le tour est joué. (*voir Annexe / Credentials pour les identifiants)*

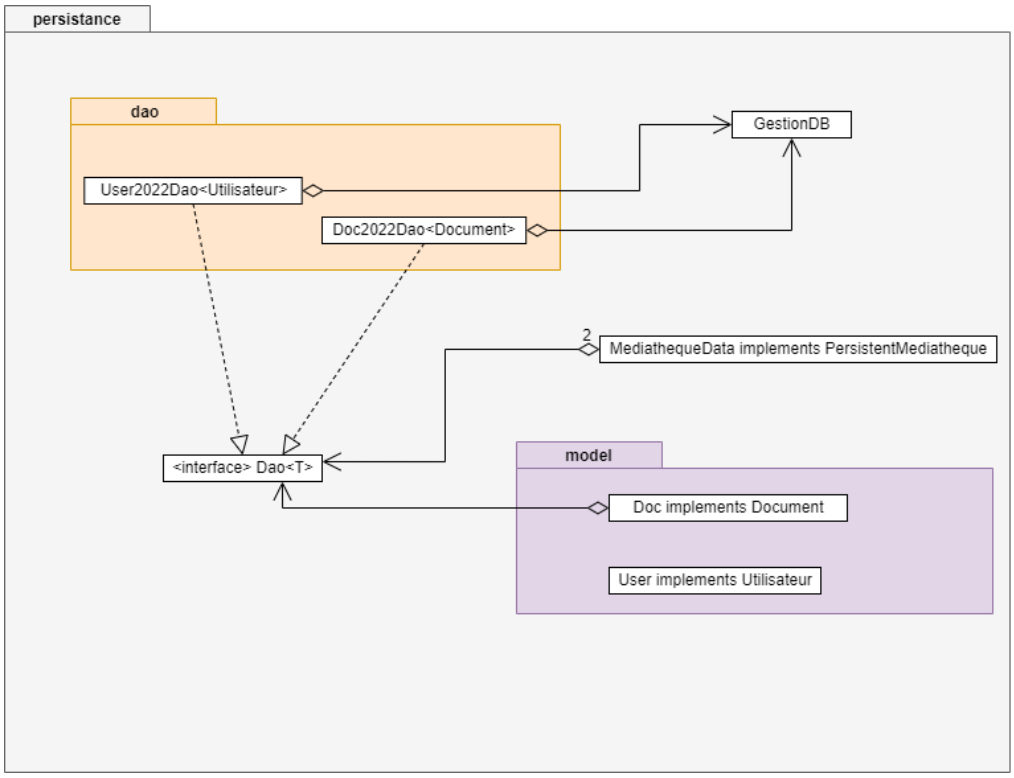
Architecture du code

Structuration du code Java

Aborescence



Package Persistance



Découplage

Comme annoncé dans le projet, le projet et scindé entre les **services**, la **meditatek2022** et la **persistance**. La persistance est elle même découplé, on retrouve d'abord les classes stables comme `MediathequeData` , `GestionDB` ou l'interface `Dao` , puis les **models** qui vont servir de representation objet d'une table dans un sgbd et implémentent **Document** et **Utilisateur** de la librairie **meditatek2022.** Enfin, les **Dao**'s (*Data Access Object*). (*Voir chapitre sur la transformation objet-relationnel*)

Injection de dépendance

Après l'injection de dépendance de `MediathequeData` et `Mediatek2022` , ce sont les Dao's qui sont injectés dans les classes. En effet, ils sont nécessaire dans leur rôle de passerelle et la mediatheque comme les documents en sont dépendant. (*Voir chapitre sur la Concurrence*)

Service et Jsp

Les services du projet sont des **Servlets** qui mettent en relation l'utilisateur avec le serveur. Couplées aux JSP's, les serlets s'occupent des requettes **Http** comme `POST` ou `GET` tandis que les jsp's représentent l'interface utilisateur et benificient d'une **pre-compilation** pour les balises jsp sur la page. Ainsi dans un **MVC**, les servlets sont les **controlleurs** et les jsp's les **vues.** Les deux communiquent avec la persistance, les servlets pour les requettes et les jsp's pour un affichage dynamique.

Servlet	Usage
Ajout	Ajouter un document à la BD
Biblio	Afficher la JSP Biblio.jsp

Servlet	Usage
Emprunt	L'utilisateur emprunte un document
Feed	Le tableau de bord de l'utilisateur
Init	Initier les singletons et le JDBC
Login	Authentifier l'utilisateur
LogOut	Deconnecter l'utilisateur
Retour	Retourner un document

JSP	Usage
Index	Racine du projet
Biblio	Interface pour bibliothecaire
Feed	Interface pour client
Login	Interface d'authentification

Transformation object-relationnel

Les DAO (Data Acces Object)

C'est un design pattern implementé pour gérer la persistance des donnés. Ils isolent la partie **application** Java de la partie **perstitante** comme la BD. Dans notre cas ils sont la **passerelle** entre les objets Java et table de la DB. L'utilisation est simple, on crée une **interface** avec les méthodes représentant les requetes élémentaires du **SQL** (**Get**, **Create**, **Update**, etc..). On a donc notre interface `Dao<T>` , il suffit maintenant de l'implémenter pour l'objet que l'on souhaite représenter, par exemple les classes :

```
public class Doc2022Dao implements Dao<Document>
public class User2022Dao implements Dao<Utilisateur>
```

Les modèles

Ce sont eux la partie **Application** Java. C'est la classe qui sera crée à partir des données persistante et des Dao. Mediatek2022 fournit les interfaces `Document` et `Utilisateur` qui après implémentation permettent d'effectuer la transformation objet-relationnel.

Dans le projet ce sont les classes `User` et `Doc`

Variables sessions

Gestion de la session de l'utilisateur

Dans le projet, on désigne trois type d'utilisateur:

- Anonyme → Utilisateur non authentifié.
- Client → Utilisateur authentifié et client dans la mediathèque, il peut emprunter et retourner des document.
- Bibliothecaire → Utilisateur authentifié et employé de la mediathèque, il ajoute les documents.

Ainsi, à chaque authentification, une session est ouverte avec l'ajout de l'utilisateur qui porte les informations sur s'il est client ou bibliothecaire.

Sécurité et accès aux services

Il est évident qu'un utilisateur non authentifié de devrait pas pouvoir accéder aux interfaces déstinés aux utilisateurs authentifié, tout comme un client ne devrait pas pouvoir ajouter des documents ou encore que le bibliothecaire ne puissent pas emprunter des documents sans compte client.

C'est pourquoi chaque services vérifient les caracteristiques de l'utilisateur et le redirige si besoin.

Service | JSP	Routage	Condition d'accès | Action sur session	HTTP
Index	mediatekProjet/	Aucune	GET
Login	mediatekProjet/login	Ne pas être authentifié	GET/POST
Feed	mediatekProjet/feed	Etre authentifié client	GET/POST
Biblio	mediatekProjet/biblio	Etre authentifié bibliothecaire	GET/POST
LogOut	mediatekProjet/logout	Libere la Session	GET
Retour	mediatekProjet/retour	Etre authentifié client	POST
Emprunt	mediatekProjet/emprunt	Etre authentifié client	POST
Ajout	mediatekProjet/ajout	Etre authentifié bibliothecaire	POST

> 💡 Si un utilisateur authentifié revient après avoir fermé sa page Mediatek et que la session est toujours ouverte, il sera redirigé vers l'interface adéquate (Feed ou Biblio). Un utilisateur non authentifié sera toujours redirigé vers l'interface d'authentification (Login)

Concurrence

La **concurrence** désigne la capacité du programme à s'éxecuter sans que ses données ou son état soient alterés, ce qui nuirait au bon fonctionnement du programme.

Ici, la **concurrence** du projet se consacre uniquement sur la gestion **thread-safe** de ses objets. Ainsi, la **concurrence** de la base de donnée est mise de côté car c'est une gestion déjà mise en place par les librairies **JDBC** (*Java DataBase Connectivity*). C'est pourquoi, la seule classe dont il est nécessaire d'assurer la persistance est la classe `Document`, car c'est la seul qui est soumise à des changements (la classe `Utilisateur` n'est pas soumise à de possible changement dans le projet). On peut en effet mettre à jour le statut de l'emprunteur d'un document, c'est pourquoi il est necessaire de `synchronized` l'objet relationnel sur lequel on effectue des changements. Par exemple les méthodes `emprunt` et `retour` de la classe `Doc`.

Mention honorable pour les **singletons**. Dans le projet, les classes `MediathequeData` et `GestionDB` representent une seule et unique **instance**. Initialisées dans un bloc static, ces classes sont alors dites thread-safe, car il est impossible d'avoir deux instances differentes de l'objet. Contrairement aux méthodes appelés `Lazy Singleton` ou `If Statement Singleton` , qui pour la première ne respecte pas la thread-safety et l'autre n'utilise pas les avantages du **Java.**

La base de donnée

Fonctionnement

Dédicasse à Julien Comoli

La BD est faite avec `MariaDB` sur `PhpMyAdmin` et est hebergé sur un serveur par mon camarade ***Julien Comoli*** en **DUT APP.**

Pour faire simple, il nous fournit un utilisateur PhpMyAdmin et cela nous permet de gérer la BD depuis le **web** (sans application tierce) et **sans hébergement local**.

Les tables (Script)

```
create table utilisateur(
    idUtilisateur int auto_increment,
    login varchar(30) not null,
    password varchar(30) not null,
    isBibliothecaire boolean default false,
    primary key(idUtilisateur),
    UNIQUE(login)
);


create table document(
    idDoc int auto_increment,
    emprunteur varchar(30),
    typeDoc int not null,
    titre varchar(30),
    auteur varchar(30),
    classification varchar(30),
    artiste varchar(30),
    style varchar(30),
    realisateur varchar(30),
    primary key(idDoc),
    foreign key(emprunteur) references utilisateur(login)
);
```

Pour simplifier les choses, l'utilisateur n'a que **3** attributs, donc le **stricte minumum** pour le projet et la méthode de **l'héritage ascendant** a été choisit pour les documents. C'est-à-dire que tout les attributs sont regroupés dans une même table et il n'y a que l'attribut `typeDoc` pour différencier un document d'un autre. A savoir que cette à méthode a été préféré à la **méthode descendante** (une table **mère** avec les attributs **redondant** et les tables **filles** et les attributs **spécifique**), car quand bien même on perd en **maintenabilité** dans la BD (regroupement infini d'attribut) c'est un **gain** et **temps** et de **lisibilité** dans le code des DAOs, car tout est regroupé dans la même table on a plus qu'à ce servir.

Efficacité

Comme dis précedemment, l'architecture des tables a été concue pour etre le plus efficace dans le code. Mais ce n'est pas le seul argument.

Le singleton `GestionDB`

Ce singleton représente une unique instance de connection. C'est-à-dire qu'une fois la connection établie dans le `InitListenner` emballé c'est pesé.

```
GestionDB.getInstance().getConnection();
```

Les requêtes et précompilation

La liste est courte mes voici toutes les requêtes du projet :

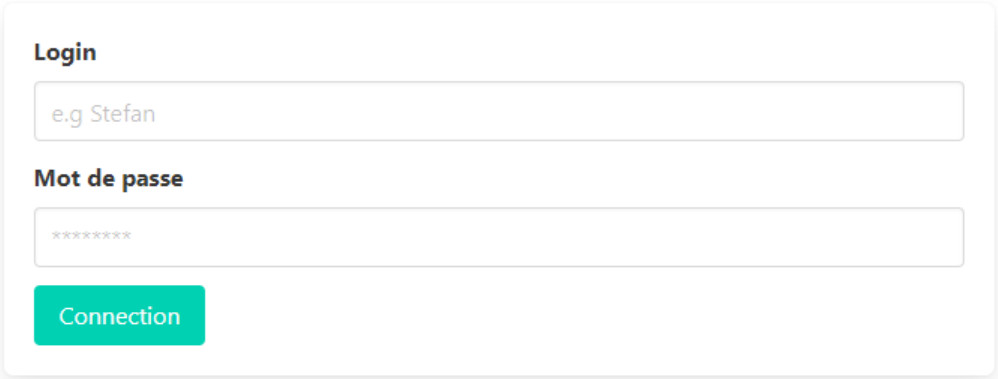
```
// Document

select * from document where idDoc = ?
select * from document
insert into document (typeDoc, titre, auteur, genre) values (?, ?, ?, ?) //livre
insert into document (typeDoc, titre, artiste, style) values (?, ?, ?, ?) //cd
insert into document (typeDoc, titre, realisateur) values (?, ?, ?) //Dvd

// Utilisateur
select * from utilisateur where login = ? and password = ?
```

Annexe

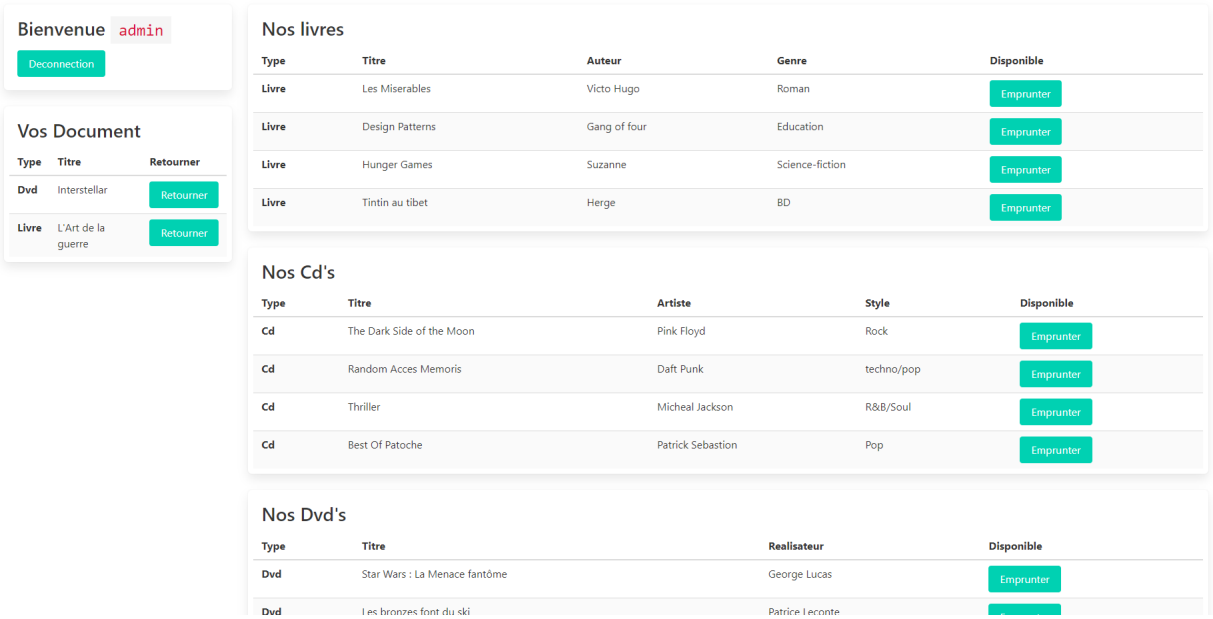
Les pages sur le web 😉

Comme dit tout au début, ce projet nécessite aucune installation particulière de SGDB, mais dans le cas où vous n'avez pas le temps de jeter un coup d'oeil et de testez vous même (très dommage ☹) j'ai screen les pages web (je sais c'est pas bien les screens):

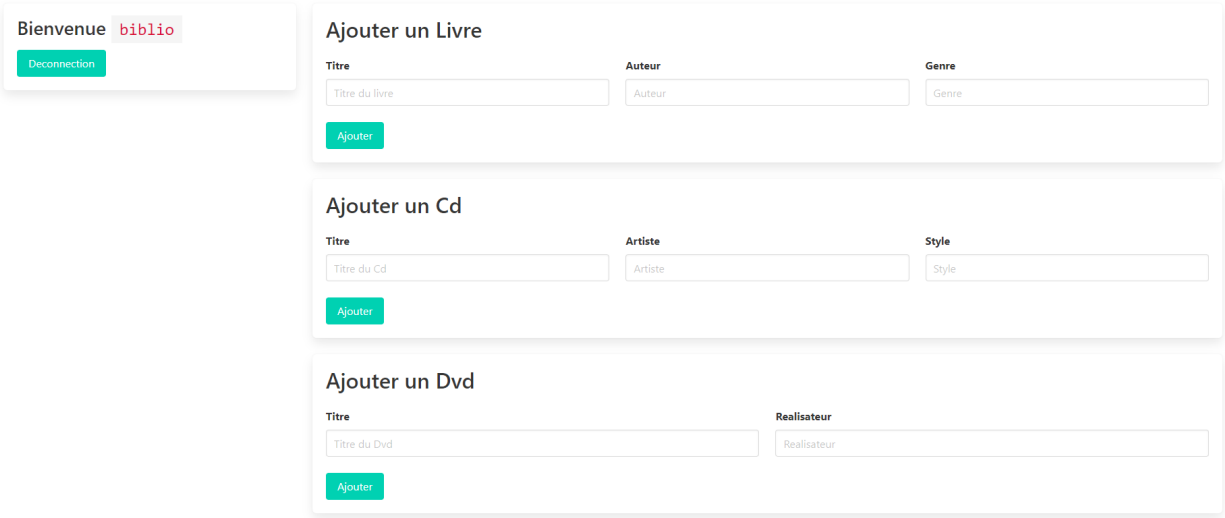
Index



Login



Feed Client



Bibliothecaire

Fait avec le framework *css Bulma* pour avoir un rendu joli rapidement.

Credentials

Client

- login → `admin`
- password → `123456`

Bibliothecaire

- login → `biblio`
- password → `123456`