Programmation répartie

Rapport projet

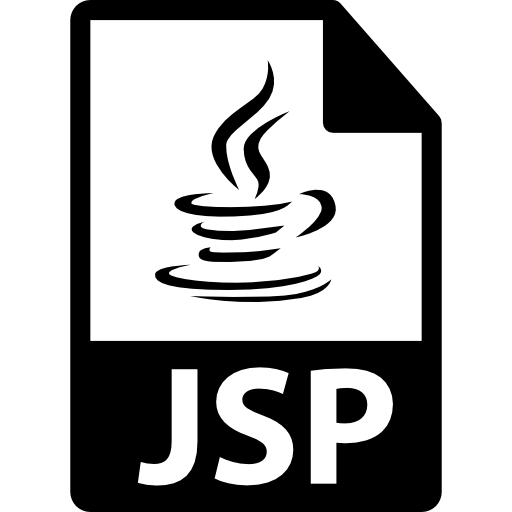


Table des matières

[Problématique : 3](#_Toc99293208)

[Structure du code : 3](#_Toc99293209)

[Implementation du module service : 4](#_Toc99293210)

[Transformation objet-relationnel : 4](#_Toc99293211)

[Utilisation des variables sessions : 5](#_Toc99293212)

[Gestion de la concurrence : 6](#_Toc99293213)

[Gestion de l’efficience de la base de données : 6](#_Toc99293214)

# Problématique :

Le but de ce projet était de réaliser une application java serveur utilisant les modules de servlet et de JSP de java EE.

# Structure du code :

L’application est décomposée en trois parties distinctes :

* Une première partie intitulé les services, elle gère tout la partie serveur et contient le code JSP et les servlets c’est elle gère l’interaction avec l’utilisateur et lui est présenté.
* Une deuxième partie intitulée mediatek2022.jar, il s’agit d’une bibliothèque qui nous est fournis avec le sujet du projet que l’on doit utiliser pour faire interagir les services et les données, en effet la première et troisième partie son strictement indépendante et nous doivent présenter aucun lien direct. Il nous est strictement interdit de modifier le code de Mediatek2022.jar ainsi cette librairie est directement importer en temps que librairie externe dans le lib de Tomcat.
* Une troisième partie intitulé persistance qui contient les interactions avec la base de données ainsi que la transformation des données java en données SQL.

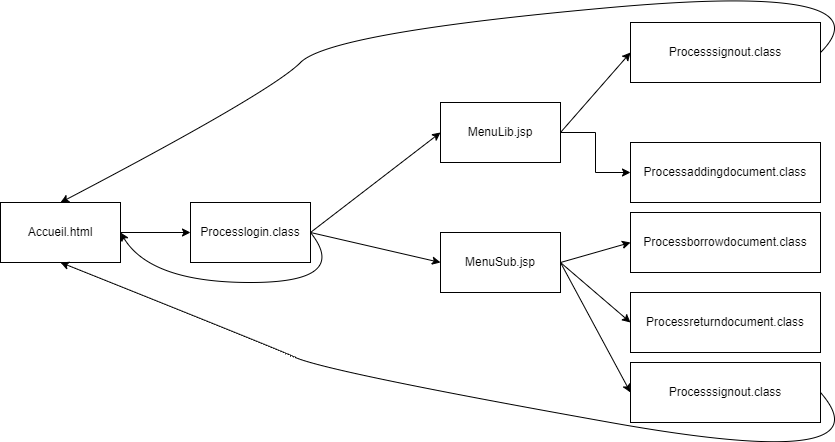
# Implémentation du module service :

J’ai choisi de crée un module service à base de fichier JSP et de Servlet java.

Utilisant des formulaires HTML pour gérer les interactions avec l’utilisateur de l’application t’utilisation de fichier JSP est idéal pour gérer ce double aspect affichage/gestion en arrière-plan de l’interaction. Les servlets sont parfaites elles pour le process des actions demandé par l’utilisateur.

L’utilisateur est accueilli par une première page html qui revoie via l’attribut action de la balise HTML form vers une servlet de traitement qui ensuite redirige l’utilisation et ainsi de suite toute au long du parcours

La structure est donc la suivante :



# Transformation objet-relationnel :

Pour permettre la persistance des données, il est nécessaire que l’application puisse stoker les objets java crée pendant sont cycle dans la base de données.

Pour transformer un objet java en un champ d’une table les informations nécessaires à l’ajout SQL sont stocké dans les attributs de l’objet comme ici :

Public synchronized void ajoutDocument(int *type*, Object... *args*) {

*// args[0] -> le titre*

*// args [1] --> l'auteur*

*// etc... variable suivant le type de document*

                ResultSet tableResult = bdd.getInsert("document (Namedoc , Typedoc)", "'" + (String) *args*[0] + "','" + *type* + "'");

        }

Dans le cas de la transformation d’une table SQL en un objets JAVA un switch sur le type d’objet permet de crée le bon objet par exemple :

ResultSet ResultDocument = bdd.getselect("document", "(Numdoc = '" + *numDocument* + "')");

                if(!ResultDocument.next()){

                    System.out.println("Request empty : No such document");

                    return null;

                }else{

                    System.out.println("There is a result : document found");

                    switch(ResultDocument.getString("Typedoc")){

                        case "1":

                            return new Book(ResultDocument.getString("Namedoc"));

                        case "2":

                            return new DVD(ResultDocument.getString("Namedoc"));

                        case "3":

                            return new CD(ResultDocument.getString("Namedoc"));

                        default:

                            throw new DatabaseProblemException("The type of the document is unknown");

                    }

                }

# Utilisation des variables sessions :

La partie service utilise des variables sessions pour enregistrer des informations :

* L’utilisateur connecter qui contient tous les documents emprunté et qui est donc mis à jour après chaque emprunt et chaque retour.
* Le login et le mot de passe que l’utilisateur à entrer afin de permettre de l’actualiser.

Ces trois variables sont initialisées dans Processlogin.class en cas de connexion réussi et sont utilisé durant tout le cycle de vie des services.

# Gestion de la concurrence :

L’application étant une application web utilisant un serveur elle doit pouvoir gérer la concurrence des ressources partagé.

Pour cela des verrou sont placé dans l’application aux endroits utilisant des ressources partagé.

Dans les classes java

Public synchronized Document getDocument(int *numDocument*)

Ainsi que dans la partie Service

HttpSession TheSession = request.getSession(true);

    Synchronized (TheSession) {

# Gestion de l’efficience de la base de données :

L’application est configurée pour fonctionner avec le SGBD MariaDB.

Un fichier Preparedatabase .java contient toute les modèles de requêtes et effectuant la création des statement et des résultat.

Lorsque l’application a besoin d’accéder à une ressources de la base de données elle n’a qu’à faire appel à cette classe