

SOLUTION FONCTIONNELLE ET TECHNIQUE

AMELIOREZ LE SYSTEME D'INFORMATION DE LA BIBLIOTHEQUE



I – LE DIAGRAMME DE CLASSES UML

II – LES PRINCIPALES REGLES DE GESTION

III – LE MODELE PHYSIQUE DE DONNEES (MPD)

IV – LA SOLUTION TECHNIQUE MISE EN PLACE

V – LA MISE EN OEUVRE DU SYSTÈME

Mise en place des modules Maven

1. Web service.
2. Côté client.

Ajout des dépendances dans les fichiers pom.xml

Mise en place du serveur Glassfish

Configuration du serveur Tomcat dans l'IDE

Création d'un fichier jar exécutable pour le batch

VI – UTILISATION DE GIT

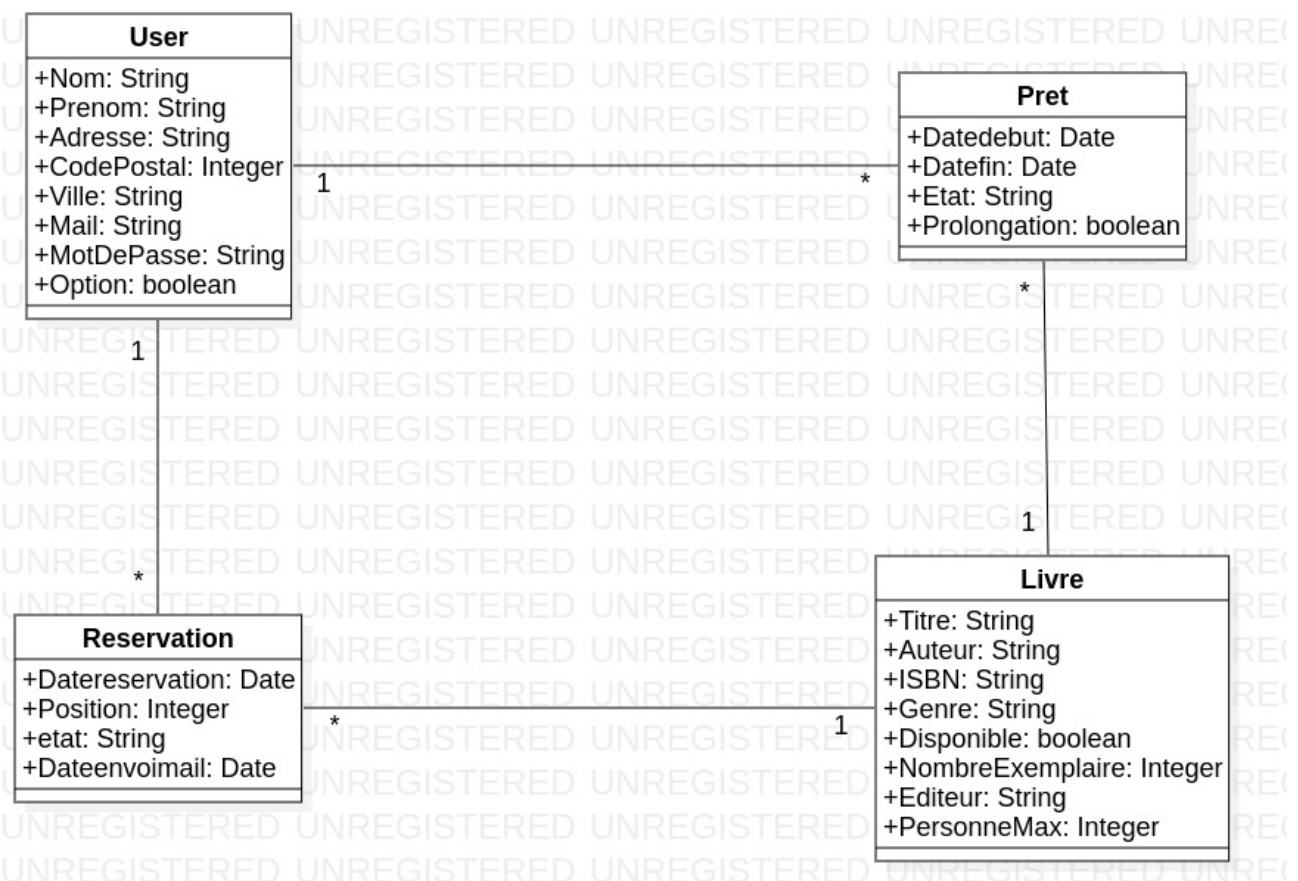
Création des branches

I - Le diagramme de classes UML.

Pour cette amélioration du système, j'ai ajouté au diagramme UML une classe reservation avec les attributs suivants :

- date de réservation
- la position
- l'état de la réservation
- la date d'envoi du mail

Et l'ajout des attributs option dans la classe User et personnemax dans la classe Livre.



II - Les principales règles de gestion.

➔ La relation de la classe User et de la classe Pret :

L'utilisateur peut faire plusieurs prêts et il y a un utilisateur par prêt.
Relation 1 - *

➔ La relation de la classe Pret et de la classe Livre :

Plusieurs livres peuvent être empruntés. Un prêt d'utilisateur est lié à un seul livre.
Relation 1 - *

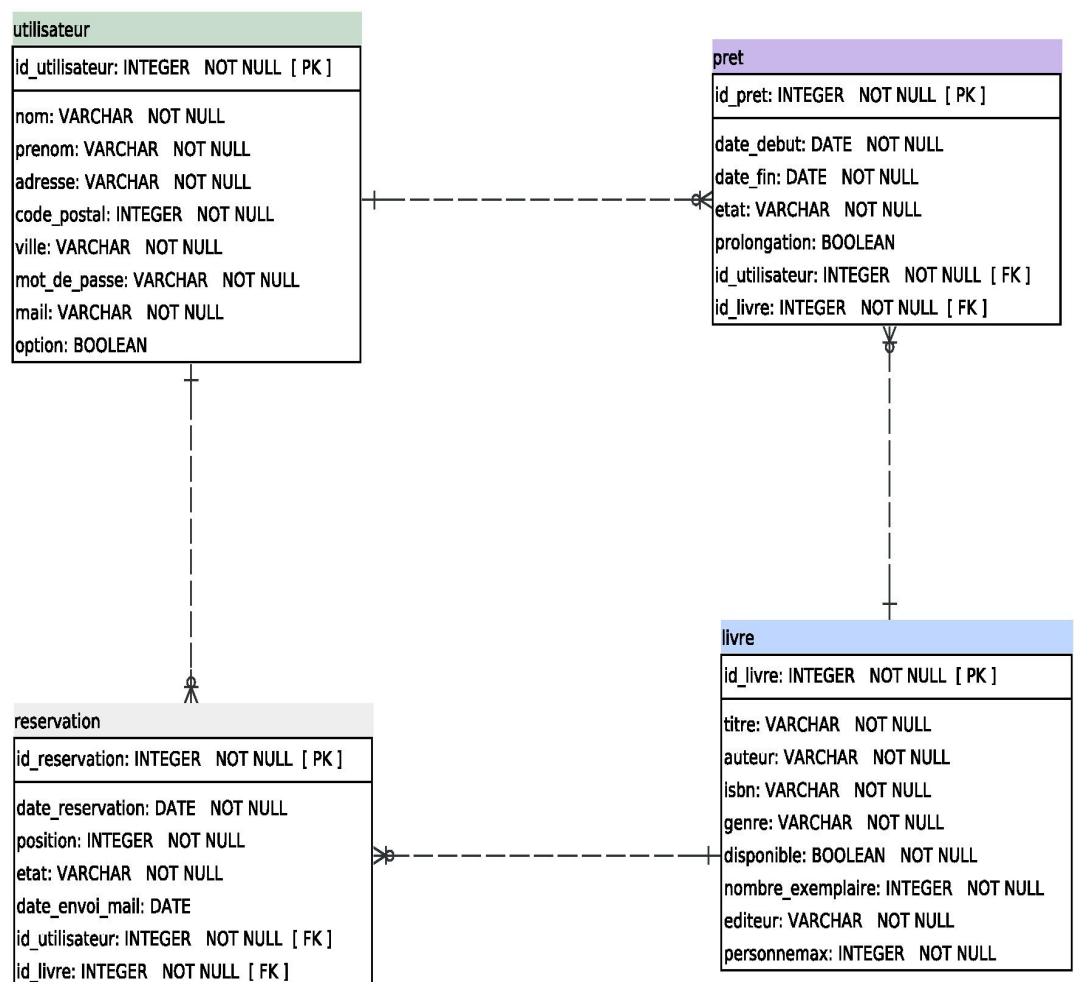
➔ La relation de la classe User et de la classe Reservation :

Un utilisateur peut faire une ou plusieurs réservations.

➔ La relation de la classe Reservation et de la classe Livre :

Une ou plusieurs réservation pour un livre.

III - Le modèle physique de données (MPD).



IV - La solution technique.

Etape 1 :

Configuration de la base de données

Mise en place de la base de données db_library et insertion des données dans les différentes tables.

Etape 2 :

Configuration du serveur d'application Glassfish 5

Installation et lancement du serveur.

Etape 3 :

Configuration d'Apache Maven

J'ai réalisé le squelette de l'application de prêts de livres pour une bibliothèque avec Apache Maven. J'ai créé deux projets Maven, un pour le web service et un autre pour le client.

Etape 4 :

Côté web service

J'ai mis en place 4 modules (business, consumer, model et webapp). J'ai également utilisé le framework String et String Jdbc pour communiquer avec la base de données. Et JAX-WS qui permettra d'exposer les méthodes dans le web service.

Etape 5 :

Côté client

Pour le côté client, j'ai mis en place 5 modules (batch, business, consumer, model et webapp).

Le module batch servira pour l'envoi de mail pour la relance des prêts non rendus à temps. J'ai utilisé le framework String et le framework Struts pour la gestion de l'affichage du site web.

Etape 6 :

Déploiement des services sur Glassfish et création d'un web service

Dans la fenêtre du serveur Glassfish, on clique sur Applications et Deploy. On peut alors déployer les services.

V - La mise en oeuvre du système.

Mise en place des modules Maven :

1- Côté web service

Création du projet en ligne de commande :

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.libraryservice -DartifactId=libraryservice -Dversion=1.0-SNAPSHOT
```

Création des modules

module : libraryservice-webapp

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-webapp -DgroupId=org.projet.libraryservice -  
DartifactId=libraryservice-webapp -Dpackage=org.projet.libraryservice.webapp
```

Création des méthodes du web service. Exemple : PretsWS.java

module : libraryservice-business

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.libraryservice -DartifactId=libraryservice-business -  
Dpackage=org.projet.libraryservice.business
```

module : libraryservice-consumer

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.libraryservice -DartifactId=libraryservice-consumer -  
Dpackage=org.projet.libraryservice.consumer
```

module : libraryservice-model

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.libraryservice -DartifactId=libraryservice-model -  
Dpackage=org.projet.libraryservice.model
```

V - La mise en oeuvre du système.

2- Côté client

```
mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.library -DartifactId=library -Dversion=1.0-SNAPSHOT
```

module : library-batch

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.library -DartifactId=library-batch -Dpackage=org.projet.library.batch
```

module : library-webapp

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-webapp -DgroupId=org.projet.library -  
DartifactId=library-webapp -Dpackage=org.projet.library.webapp
```

module : library-business

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.library -DartifactId=library-business -  
Dpackage=org.projet.library.business
```

module : library-consumer

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.library -DartifactId=library-consumer -  
Dpackage=org.projet.library.consumer
```

Le consumer.impl appelle la méthode du web service.

module : library-model

```
mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -  
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1 -  
DgroupId=org.projet.library -DartifactId=library-model -Dpackage=org.projet.library.model
```

Le module model va récupérer les fichiers des méthodes du web service.

V - La mise en oeuvre du système.

3 - Ajout des dépendances dans les fichiers pom.xml

4 - Mise en place du serveur Glassfish :

- ✓ Installation du serveur Glassfish.
- ✓ Lancement de la commande pour démarrer le serveur :
'/home/fred/Bureau/glassfish5.0/glassfish5/bin/asadmin' start-domain mydomain
- ✓ Saisir l'URL <http://localhost:4848> dans un navigateur web.
- ✓ Récupération du fichier libraryservice-webapp.war.
- ✓ Récupération du nom de la méthode du service, génération du WSDL décrivant le service.
- ✓ Test du service.

Génération des classes dans le module model client.

La génération est faite en ligne de commande pour chaque service.

- ✓ Livres (liste, disponibles, recherche)

```
[fred@linux library-model]$ wsimport -Xnocompile -d ./src/main/java -p  
org.projet.library.model.livre http://linux.home:8080/libraryservice-  
webapp427024690506988823/LivreWS?wsdl
```

- ✓ Prêts de livres

```
[fred@linux library-model]$ wsimport -Xnocompile -d ./src/main/java -p  
org.projet.library.model.prets http://linux.home:8080/libraryservice-  
webapp427024690506988823/PretsWS?wsdl
```

- ✓ Login utilisateur

```
[fred@linux library-model]$ wsimport -Xnocompile -d ./src/main/java -p  
org.projet.library.model.user http://linux.home:8080/libraryservice-  
webapp427024690506988823/UserWS?wsdl
```

- ✓ Réservations des livres

```
[fred@linux library-model]$ wsimport -Xnocompile -d ./src/main/java/ -p  
org.projet.library.model.reservations http://linux.home:8080/libraryservice-  
webapp427024690506988823/ReservationWS?wsdl
```

V - La mise en oeuvre du système.

5 - Configuration du serveur Tomcat dans l'IDE

La configuration du serveur s'effectue dans l'IDE Eclipse ;

Configuration du path ;

Configuration du Server Runtime Environnement : Apache Tomcat v8.5 et récupération du fichier d'installation Tomcat à la racine du disque dur de l'ordinateur.

Le serveur est déployé en local sur le port :

<http://localhost:8081/library-webapp/index.action>

6 - Création d'un fichier jar exécutable pour le batch

Le fichier jar : library-batch-1.0-SNAPSHOT.jar. Le batch envoie un mail de relance tous les jours avec la configuration du `cron = "0 */1 * * * ?"`

VI – UTILISATION DE GIT

Création des branches

`git branch enhancement`

Une branche enhancement pour l'ajout d'un système de réservation d'ouvrage.

`git branch bug`

Une branche bug pour la correction du bug des prolongations des livres.

Basculer sur une branche et travailler dessus

`git checkout nom de la branche`

Pusher la branche vers le repository distant

`git push origin nom de la branche`

`git status`

`git add fichier`

`git commit -m`

`git push origin nom de la branche`