Projet n° 7

**Conception d’un système d’information**

**de la bibliothèque d’une ville**

*Une image contenant clipart

Description générée automatiquement*

Documentation générale

**Habib BAH**

Parcours Développeur d’Application java

Sommaire

1. Cahier des charges
2. Architecture Web
3. Diagramme de classe
4. Web Service
5. Client SAOP
6. Batch

**Cahier de charge**

Nous souhaitons mettre en place le système d’information de la bibliothèque d’une grande ville.

L’objectif est de mettre en place une application web qui permet aux utilisateurs de faire toutes les démarches liées à la réservation d’un livre sans se déplacer pour cela.

L’application devra permettre entre autres de :

* Rechercher des ouvrages et voir le nombre d’exemplaires disponibles.
* De suivre leurs prêts encours. Les prêts sont pour une période de 4 semaines.
* De prolonger un prêt.
* Un espace spécifique pour le personnel leur permettant de faire la gestion sur l’ensemble des emprunts et autre chose.
* Un batch qui envoie un mail aux utilisateurs pour leur rappeler de rendre leurs livres dans les temps impartis.

Pour ce faire, nous avons mis en place un web service, une application web qui se connecte au web service et un batch qui s’occupe de l’envoie des mails aux utilisateurs.

**Architecture web**



Ce diagramme nous montre comment fonctionne notre système de manière globale :

1. Le client demande au stub de faire appel à un service.
2. Le stub se connecte au SKELETON et lui envoie la requête SAOP
3. Le SKELETON fait appel au service du Web Service
4. Le Web Service retourne la réponse au SKELETON
5. Le SKELETON envoie le résultat dans une réponse SAOP au stub
6. Le stub fournie le résultat au client

**Le stub est une classe qui se situe côté client et le skeleton est son homologue coté serveur. Ces deux classes se chargent d'assurer tous les mécanismes d'appel, de communication, d'exécution, de renvoi et de réception du résultat.**

**Diagramme de classe**



Nous avons fait simple pour le diagramme de classe.

* La classe utilisateurs, représente un utilisateurs avec tous ses attributs.
* La classe livre, représente un livre. Nous avons choisi de mettre en avant son titre, son auteur, le nombre de pages qu’il contient et le nombre d’exemplaires disponible.
* La classe réservation, qui présente un prêt. Nous avons décidé de ne retenir que le titre du livre, le nom et prénom de l’utilisateur, son adresse mail, la date de début du prêt et la date de fin du prêt.

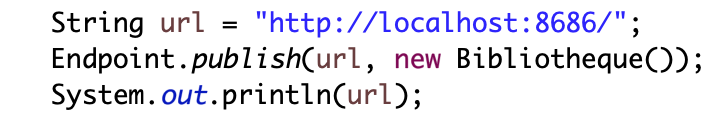
**Web Service**

Une image contenant capture d’écran

Description générée automatiquement

Le Web Service qu’on a appelé ‘BibliothqueService’, contient un certain nombre de méthodes qui traitent l’ensemble des opérations que pourrons effectuer les utilisateurs mais aussi le personnel sur l’application.

* Les balises <message name = ‘….’> </message> correspondes aux methoses (ou services) que fournit le web service.
* La balise <service name=’…’></service> Definit le nom de notre web service.



Nous avons créé le stub nous lui avons donné une adresse afin de générer le wsdl.

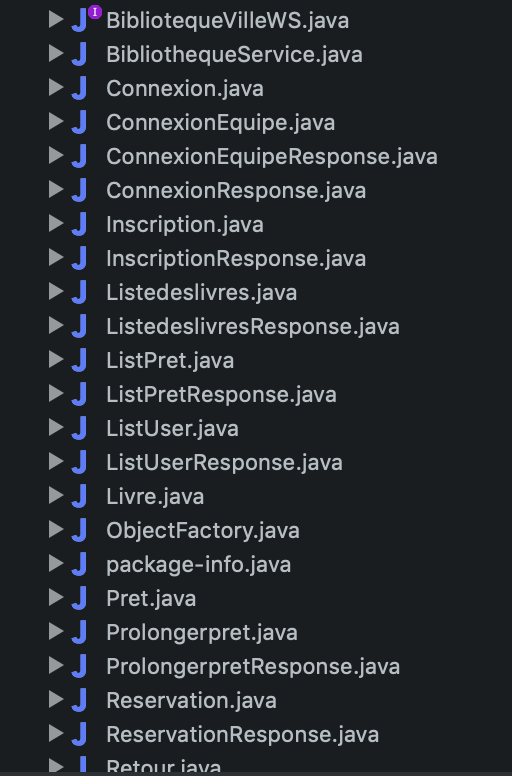
C’est grâce à ce stub aussi appelé serveur proxy que nous pourrons accéder au web service grâce à l’adresse qu’on a donné à notre serveur en l’occurrence ‘http://localhost/8686’. A cette adresse il faudra coller le wsdl à fin qu’il puisse trouver nos web méthode.

L’adresse devient donc : <http://localhost:8686/BibliotequeVilleWS?wsdl>

**Client SAOP**

Maintenant que nous avons notre web service, avec son serveur proxy (stub), il nous faut l’importer chez le client à fin qu’il puisse y avoir accès.

Nous utilisons donc la commande ‘wsimport -keep -p ’ suivi de l’adresse de notre wsdl afin de récupérer tous les packages de notre web service dans notre client.



Voilà le résultat de la commande. Nous avons toutes les méthodes de notre web services qui s’importent sur notre client. Nous pouvons donc les utiliser.

A ce stade, la démarche consiste à mettre en place un Controller, des vues car nous partons sur une architecture MVC : Model vue Controller

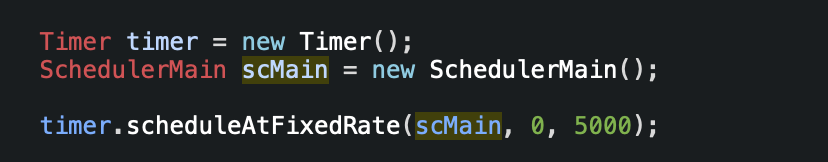
* **Le Controller** qui a pour rôle de faire un aiguillage pour le traitement des requêtes des utilisateurs.
* **Le model** dans lequel le traitement est effectué
* **La vue** qui retourne une vue à l’utilisateur

**Batch**

Pour faire des relances aux utilisateurs, nous avons mis en place un service de relance régulière qui envoi des mails aux usagers pour leur rappeler la date de fin de leur prêt pour ceux qui sont encore dans les temps, et pour ceux qui sont en retard leur rappeler qu’ils sont en retard.

Pour faire ses taches, nous avons choisi le module Schedule qui est simple à mettre en place.

Ce module étends la classe TimerTask et nous faisons appel à sa méthode scheduleAtFixedRate à fin de définir fréquence d’exécution de la tâche.

****

Sur cet exemple, nous avons défini une tache qui s’exécute toute les 5 secondes.