Documentation

Présentation générale

Le projet 4 est une application web permettant à des utilisateurs de gérer leurs emprunts dans une bibliothèque. Ils peuvent ainsi effectuer les actions suivantes :

- Consulter la liste des livres de la bibliothèque
- Se créer un compte
- Rechercher un livre par mot-clé
- Emprunter un livre s'il est disponible
- Consulter les dates de rendu des livres empruntés sur son profil
- Prolonger une fois un emprunt

Pour répondre à ces besoins, une application web en plusieurs parties a été développée.

On distingue ainsi :

- Un web-service pour réaliser le traitement des données et communiquer avec la base de données.
- Une web-app pour afficher les vues sur le navigateur de l'utilisateur.
- Un batch qui vérifie que les ouvrages ont bien été rendus et envoi un mail à l'utilisateur si il a oublié de rendre un ouvrage avant la date limite d'emprunt.

Ces trois parties communiqueront par protocole SOAP. On va donc générer un WSDL qui va permettre la communication entre chacune.

La web-app ainsi que le batch communiquent avec le web-service.

Règles de gestion :

Les règles citées ci-dessous ne sont pas forcément les plus pertinentes en pratique, mais en l'absence d'un cahier des charges et de la formalisation d'un besoin précis, il a été décidé de choisir arbitrairement des règles un minimum crédibles mais limitant le temps de développement de l'application.

Un utilisateur peut consulter la liste des livres de la bibliothèque même si il n'a pas de compte utilisateur. Il est pour cela aidé d'un outil de recherche par mots-clés.

Pour emprunter un livre, l'utilisateur doit posséder un compte utilisateur qu'il aura créé au préalable en renseignant un pseudo, un mot-de passe et une adresse email.

Deux utilisateurs ne peuvent pas avoir un même pseudo.

Il peut exister plusieurs exemplaires d'un même livre. S'il reste des exemplaires disponibles, un utilisateur peut emprunter ce livre.

L'utilisateur peut rendre un des livres qu'il a emprunté quand il le désire avant la date limite de rendu, qui est fixée deux semaines après la date d'emprunt.

Un utilisateur peut prolonger une seule fois un emprunt. Cela repousse la date de rendu de deux semaines.

Un utilisateur peut emprunter autant d'ouvrages qu'il le souhaite.

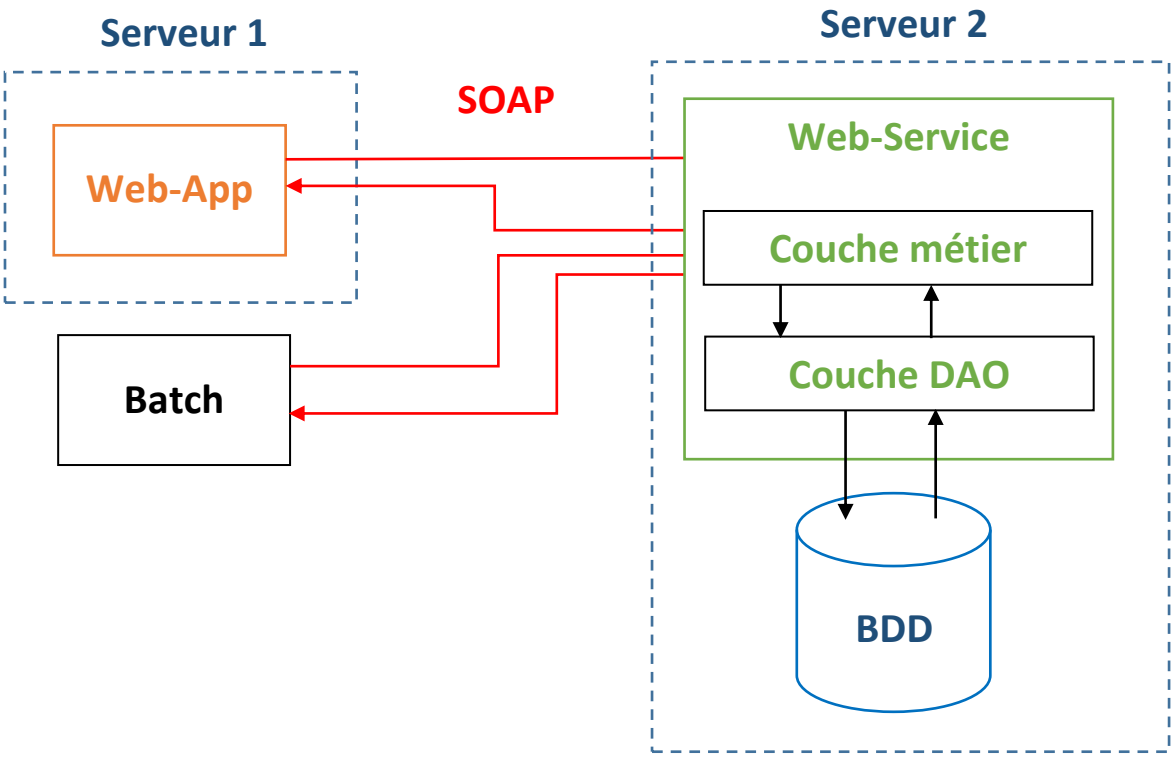
Le fait de n'avoir pas rendu un ouvrage avant la date limite n'empêche pas à l'utilisateur d'emprunter un autre livre.

Un utilisateur peut emprunter plusieurs exemplaires d'un même livre.

Diagramme de classe

Documentation technique :

Voici donc le schéma qui résume l'architecture technique globale:



Web-app

Pas grand-chose à dire ici. On a une simple web-app composée de vues jsp dont la cinématique est gérée par le framework Struts 2. Lorsque cela est nécessaire, il fait appel à des méthodes du web-service en utilisant le protocole SOAP.

Web-service

Le web-service est composé de deux couches, qui sont représentées par deux modules dans Maven.

La couche métier est composée des classes correspondant aux méthodes concernant chaque objet. On a donc les classes :

- LivreManager
- OuvrageManager
- UserManager

Dans ces classes on retrouve les méthodes qui vont effectuer les traitements nécessaires sur les données récupérée par la couche DAO. Chacune de ces classes fait donc appel à la couche DAO pour lire ou écrire dans la base de données, par le biais d'une interface. Pour découpler ces classes et instaurer une inversion de contrôle, on utilise le framework Spring pour gérer l'instanciation des classes DAO. Les méthodes appelées sont également rangées dans des classes selon l'objet qu'elles manipulent :

- UserDao
- LivreDao
- OuvrageDao

Ici, la connexion à la base de données se fait sans utiliser de pool de connexion. On créé une connexion à l'appel de chaque méthode. En production, cette technique n'est pas acceptable, il faudrait utiliser une API qui gère les pools de connexion comme BoneCP ou un framework comme Hibernate par exemple. Cependant, pour le contexte actuel, ce schéma de connexion fonctionne.

Batch

Le batch est constitué d'une classe Main qui fait appel au web-service. Cette classe fait également appel à une autre classe du batch appelée Email. Cette-dernière comporte la méthode qui assure l'envoie de l'email à l'utilisateur grâce à son adresse qui lui a été passée en paramètre. Elle utilise une API de Google et passe par une adresse gmail créée pour ce projet.

Script SQL

1° -- Création de la base de données :

```
(createdb ma_base)
createdb –U postgres ma_base
```

(Suppression de la base de données

```
dropdb ma_base)
```

2° -- Accéder à la base :

```
psql ma_base
psql –U postgres ma_base
```

3° -- Créer une table :

Table « utilisateur » :

```
CREATE TABLE utilisateur (
    id          serial primary key not null,
    pseudo      varchar(30),
    password    varchar(30),
    email       varchar(50)
);
```

Table « livre » :

```
CREATE TABLE livre (
    id serial primary key not null,
    titre           varchar(30),
    description     text,
    auteur          varchar(30),
    editeur         varchar(30),
    nb_exemplaire   int,
    nb_restant      int,
```

```
);
```

Table «ouvrage » :

```
CREATE TABLE ouvrage(
    id serial primary key not null,
    id_emprunteur   int references
utilisateur(id),
    date_emprunt    varchar(30),
    date_retour     varchar(30),
    prolongement    boolean,
    id_livre       int references livre(id),
);
```

JEU DE DONNEES :

INSERT INTO livre (titre, description, auteur, editeur, nb_exemplaire, nb_restant) VALUES ('Recettes de cuisines', 'Les meilleures recettes', 'El Cocinero', 'Gourmandise', 1, 1) ;

INSERT INTO livre (titre, description, auteur, editeur, nb_exemplaire, nb_restant) VALUES ('Lonely Planet : Bresil', 'Guide de tourisme pour le Brésil', 'Jorge Ben', 'TravelBooks', 2, 2) ;

INSERT INTO livre (titre, description, auteur, editeur, nb_exemplaire, nb_restant) VALUES ('Le Petit Prince', 'Histoire d un enfant qui tombe d une etoile et qui rencontre un renard qui lui explique la vie', 'Antoine de St-Exupery', 'Pocket', 3, 3) ;

INSERT INTO ouvrage (prolongement, id_livre) VALUES (false, 3) ; (x3)

INSERT INTO ouvrage (prolongement, id_livre) VALUES (false, 2) ; (x2)

INSERT INTO ouvrage (prolongement, id_livre) VALUES (false, 1) ; (x1)

INSERT INTO utilisateur (pseudo, password, email) VALUES (test, 123, test@gmail.com);

Configuration et déploiement

L'application se présente donc comme dit précédemment en deux parties. La web-app et le web-service. Ces deux parties sont déployées sur deux serveurs d'application différents.

On a choisi de déployer le web-service sur glassfish et la web-app sur tomcat. Assurez-vous donc d'abord d'avoir bien installé ces deux serveurs sur votre machine.

Déploiement du web-service :

Lancer dans l'invite de commande les commandes :

- cd C:\[…]\glassfish5\bin
- asadmin start-domain domain1

Ensuite allez à l'adresse suivante dans un navigateur web :

- localhost:4848

Vous devriez arriver sur la page d'accueil de Glassfish.

Cliquez ensuite sur « Application » dans le menu

Sur la page « Application », cliquez sur « Deploy » et choisissez le jar du web-service qui se trouve dans le dossier « target » du projet web-service.

Cliquez ensuite sur le lien « Launch » du web-service.

Déploiement de la web-app à partir d'Eclipse:

Importez le projet bibli-webapp dans votre workspace d'Eclipse.

Si tomcat n'est pas configuré dans le Runtime Environment d'Eclipse, cliquez sur :

Window → Servers → Runtime Environments → Add puis sélectionnez votre version de Tomcat

Ensuite, dans l'onglet « Server » en bas d'Eclipse : clic droit → Add and Remove et choisissez le projet bibli-webapp.

Assurez-vous que le server Tomcat ne tourne pas sur le même port que Glassfish. Si c'est le cas, vous pouvez changer le port de Tomcat.

Vous pouvez maintenant lancer le server Tomcat.

Ecrivez l’adresse suivante dans un navigateur web pour pouvoir enfin utiliser l’application (en adaptant le bon port):

http://localhost:*8081*/bibli-webapp/