Maxime CHAUVET & Laetitia SAVAJOL

ISIMA | F3 2015 | Jee

Développement d’un Web Service avec un client simple

rapport

Sommaire

[Introduction 2](#_Toc405754379)

[Définitions de la base de données 3](#_Toc405754380)

[Utilisation du Datastore de Google 4](#_Toc405754381)

[Le Web Service 7](#_Toc405754382)

[Client de test pour le Web Service 9](#_Toc405754383)

[Organisation de l’application 10](#_Toc405754384)

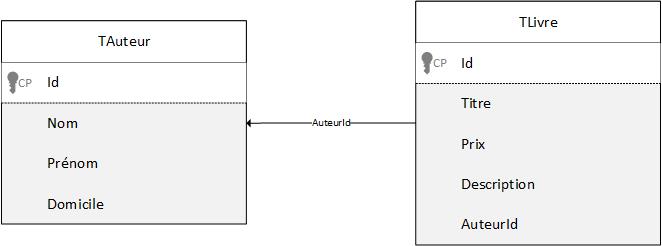
Introduction

Dans le cadre de TP de JEE, nous avons été amenés à créer un web service permettant une gestion de livres et d’auteurs. Dans cette première partie, nous avons créé un client web simple permettant d’avoir une interface de gestion complète de la base.

Dans ce rapport nous présenterons l’architecture de la base de données et de l’application. Nous ferons une présentation des technologies utilisées et pourquoi avoir choisi ces technologies. Et pour terminer, nous expliquerons le fonctionnement de l’application et comment se font les échanges avec le serveur.

Définitions de la base de données

Pour ce projet, nous sommes partis du schéma relationnel suivant. Nous avons considéré qu’un Livre est écrit par un et un seul auteur.



Ces deux tables nous sont utiles pour pouvoir développer l’application. Dans un premier temps, la table « TAuteur » nous permet de lister tous les auteurs ayant écrit des livres :

* Id : identifiant unique de l’auteur (type « long ») ;
* Nom : nom de l’auteur (type « String ») ;
* Prénom : Prénom de l’auteur (type « String ») ;
* Domicile : Lieu où habite l’auteur (type « String ») ;

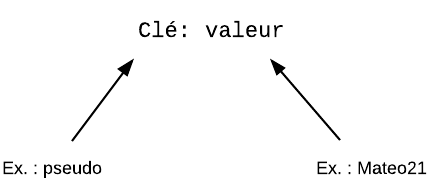
Dans un second temps, la table « TLivre » permet de lister tous les livres écris. Une liaison avec la table « TAuteur » permet de distinguer les livres écris par chaque Auteur :

* Id : identifiant unique du livre (type  « long ») ;
* Titre : titre du livre (type « String ») ;
* Prix : le prix du livre (type « double ») ;
* Description : la description du livre (type « String ») ;
* AuteurId : identifiant de l’auteur (type « long »). Cette colonne est une clé étrangère de la table de « TAuteur » ;

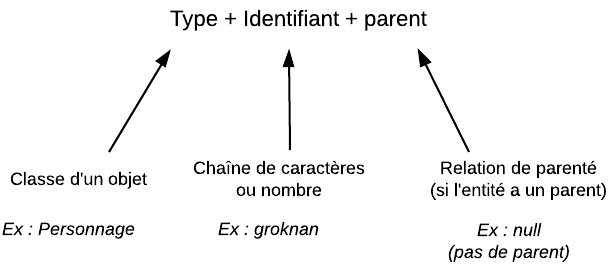
Utilisation du Datastore de Google

Pour stocker ces données, nous avons utilisé le Datastore proposé par Google. Il est gratuit, rapide, capable de gérer des fortes charges et reparti sur plusieurs Datacenter. Il a une particularité par rapport aux bases habituelle c’est que c’est une base dite NoSQL (***N****ot* ***O****nly* ***SQL***).

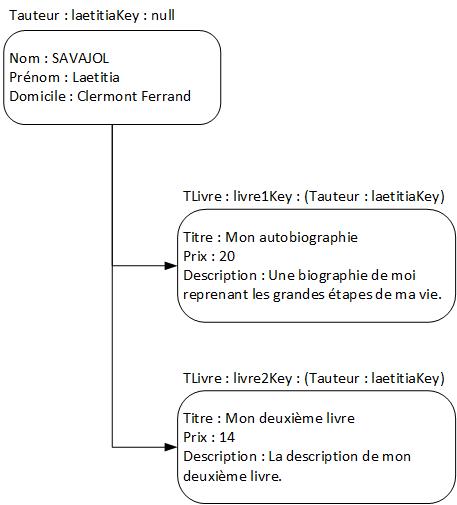
Le Datastore n’est pas une base de données relationnelle. Il n’y a pas de schéma. On dit que le Datastore est « schemaless » (sans schéma). Il fonction de manière très simple :



Le Datastore propose un système d’entité qui permet d’organiser les données. On peut les assimiler aux tables dans un schéma relationnel. Elles regroupent ainsi plusieurs paires de clé-valeur. Ainsi une entité est un objet possédant des attributs correspondant aux différentes colonnes dans une table de schéma relationnelle. Ensuite il faut pouvoir identifier une entité. Pour cela, le Datastore propose la façon suivante :



En appliquant la philosophie du Datastore proposé par Google sur le schéma relationnel de départ on obtient l’architecture suivante :



Sur chaque entité, on peut effectuer 4 opérations de base qui sont disponibles dans un base NoSQL comme le Datastore :

* Get() : récupère l’entité en fonction de l’identifiant passé en paramètre ;
* Put() : écrit (ou écrase) une entité dans la base ;
* Delete() : supprime une entité en fonction de l’identifiant passé en paramètre ;
* Query() : retrouve une entité en fonction des critères passés en paramètre ;

Pour effectuer cette notion de parent que l’on peut voir dans la hiérarchie ci-dessus, on a utilisé « l’ancestor » (ancêtre en anglais). Ainsi à la création d’un livre, on lui affecte un auteur qui sera son ancêtre comme le montre le code ci-dessous :

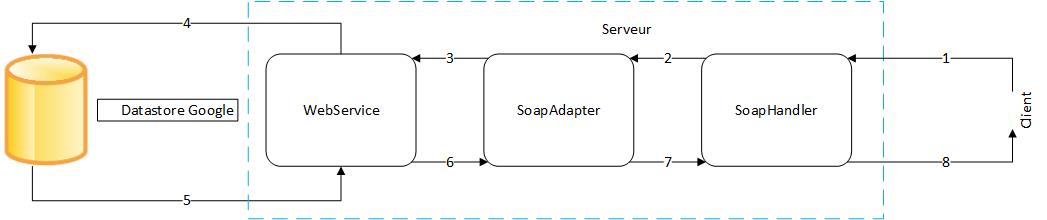
Entity monLivre = **new** Entity(ConstanteMetier.*ENTITY\_TLIVRE*, KeyFactory.*createKey*(ConstanteMetier.*ENTITY\_TAUTEUR*, auteurId));

Ensuite, l’accessions aux livres d’un auteur spécifique ce fait en utilisant la méthode « setAncestor() » dans laquelle on renseigne la clé de l’entité parente.

Query q = **new** Query(ConstanteMetier.*ENTITY\_TLIVRE*).setAncestor(KeyFactory.*createKey*(ConstanteMetier.*ENTITY\_TAUTEUR*, auteurId));

Le Web Service

Le but de ce TP était de proposer un web service permettant de gérer la base de données décrite ci-dessus. Le projet a été créé de la manière suivante :



1. Le client envoie une requête SOAP (***S****imple* ***O****bject* ***A****ccess* ***P****rotocol*) au serveur. La requête arrive dans la méthode « doPost » de la servlet en entrée du serveur. Elle fait alors au « SoapHandler ;
2. Le « SoapHandler » analyse la requête et appelle la méthode correspondant dans le « SoapAdapter » en fonction de l’entête de la requête ;
3. Le « SoapAdapter » récupère la requête et appelle la bonne méthode du Web Service avec les paramètres nécessaires ;
4. Le Web Service fait les actions utiles en base (ajout, modification, récupération ou suppression). A ce moment-là on sort du serveur ;
5. La base nous retourne alors le résultat correspondant ;
6. Le Web Service transmet lui aussi le résultat obtenu ;
7. Le « SoapAdapter » récupère le résultat du Web Service et prépare la requête de réponse ;
8. Le « SoapHandler » envoie la requête de réponse au client correspondant à la demande initiale.

Le serveur se divise en 3 modules. Le premier module, « SoapHandler » est assimilé à un gestionnaire. Il va avoir pour but de récupérer les requêtes provenant des différents clients, d’analyser leur demande et de les dispatcher vers les bonnes méthodes du Web Service. Le second module, le Web Service, lui effectue la liaison avec le Datastore de Google. Il propose toutes les méthodes CRUD (***C****reate,* ***R****ead,* ***U****pdate and* ***D****elete*) pour « TLivre » et « TAuteur ». Enfin, le dernier module, le « SoapAdapter » permet de faire la liaison entre les deux. Il est utilisé pour découpler la partie de réception de la requête et la partie de gestion des données. Ainsi, le « SoapAdapter » possèdera autant de méthode que celles dans le Web Service pour pouvoir les appeler en fonction de la demande cliente passée par le « SoapHandler » et par la suite créer la réponse reçu par le Web Service.

Client de test pour le Web Service

Pour tester le serveur Web Service, nous avons décidé de créer un client Web à l’aide de la technologie JSP. Celui-ci reprend au maximum les méthodes de gestion d’une base de données.

La technologie JSP (***J****ava* ***S****erver* ***P****ages*) fournit un moyen simple et extensible de générer du contenu dynamique pour un client Web. Une page JSP est un document texte qui décrit comment traiter la requête d'un client et comment créer une réponse. Une page JSP contient :

* Des informations de formatage (modèle) du document web, habituellement en HTML ou XML. Les concepteurs web peuvent modifier cette partie de la page sans affecter les parties dynamiques. Cette approche permet de séparer la présentation du contenu dynamique ;
* Des éléments JSP et de script pour générer le contenu dynamique du document Web. La plupart des pages JSP utilisent aussi des JavaBeans et/ou des Enterprise JavaBeans pour réaliser les opérations complexes de l'application. Les JSP permettent en standard d'instancier des beans, de modifier ou lire leurs attributs et de télécharger des applets. La technologie JSP est extensible en utilisant des balises personnalisées qui peuvent être encapsulées dans des bibliothèques de balises personnalisées (*« taglibs »*) ;

Les servlets permettent ensuite d’assurer le contrôle de l’interface côté client et de faire le lien dans un premier temps entre les différentes pages de l’application mais aussi de faire le lien avec le serveur. En effet, les servlets vont avoir le rôle d’intercepter ce que fais le client de créer les requêtes SOAP pour effectuer des demandes auprès du serveur puis de traiter les réponses et les afficher au client. Le bout de code suivant montre comment effectuer cet enchaînement :

//Création de la requête

SOAPMessage soapMessage = createRequest();

//Envoie de la requête au server et récupération de la réponse

SOAPConnectionFactory soapConnectionFactory = SOAPConnectionFactory.*newInstance*();

SOAPConnection soapConnection = soapConnectionFactory.createConnection();

SOAPMessage soapResponse = soapConnection.call(soapMessage, ConstanteMetier.*URL\_WS*);

//Traitement de la réponse

List<TAuteur> auteurs = getAllAuteur(soapResponse);

//Affichage de la liste des auteurs

req.setAttribute(ConstanteMetier.*AUTEURS\_LIST\_PARAM*, auteurs);

**this**.getServletContext().getRequestDispatcher("/WEB-INF/auteurs.jsp").forward(req, resp);

La complexité d’utiliser la technologie JSP c’est que l’on doit construire les requêtes SOAP à « la main » en JAVA. Cependant, elle permet de facilité la communication. En effet, comme on peut le voir à la fin du bout de code ci-dessus, nous pouvons passer directement des objets java à la vue en utilisant les paramètres dans le header de la requête. Ceci permet de faciliter un rendu dynamique sur l’application.

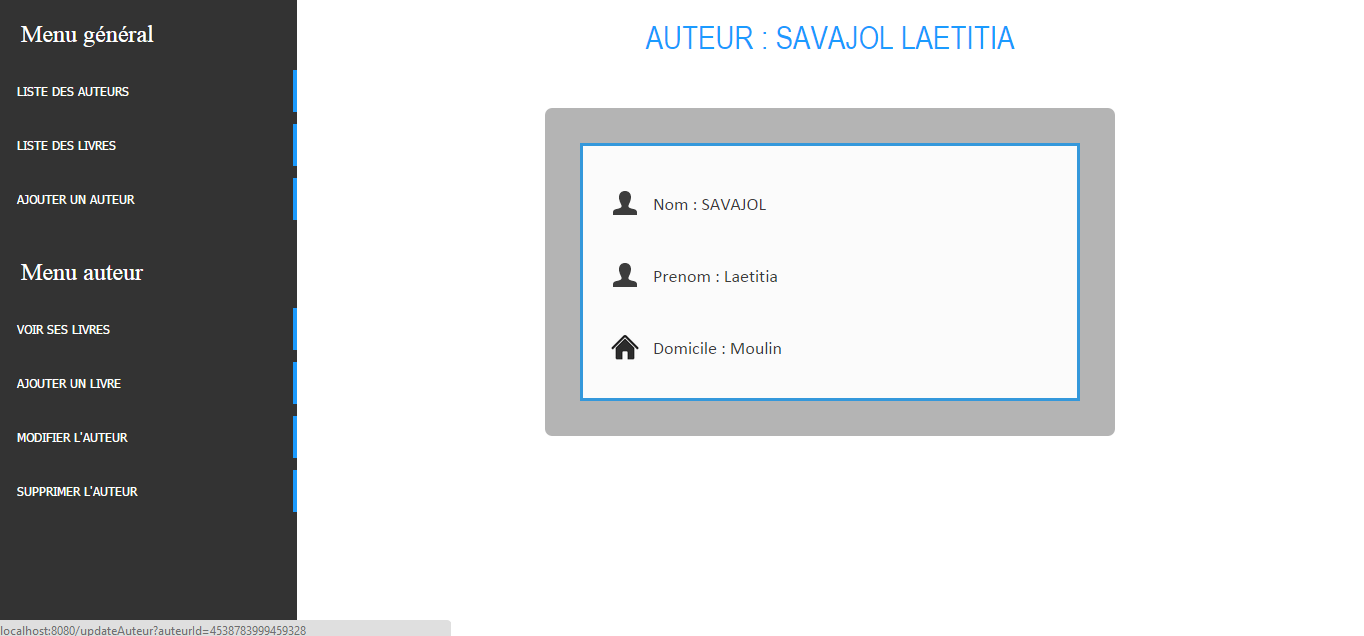
# Organisation de l’application

Afin de pouvoir exploiter au maximum cette base, nous avons conçu une application ergonomique permettant de visualiser des auteurs, d’ajouter des auteurs, les modifier mais aussi de pouvoir xvisualiser tous les livres ou la liste des livres d’un auteur en particulier. L’exemple ci-dessous montre l’organisation de l’application :



On peut voir une barre de menu sur le côté gauche de l’application. Au centre s’affichera le contenu dynamique en fonction de ce que souhaite voir ou faire le client.

Lorsque le client se trouve sur une page d’un auteur, la barre de menu évolue un peu comme on peut le voir ci-dessous :



C’est à partir de cette page que le client pour accéder aux différentes actions de bases sur les auteurs. Il pourra aussi consulter les différents livres de cet auteur en particulier et pouvoir effectuer les actions de bases sur chaque livre.

Conclusion

Tout au long de cette première partie, nous avons été amenés à travailler sur les nouvelles technologies proposées par le JEE. Cela nous a permis de découvrir la puissance de développer une application côté serveur en proposant un web service à tous les utilisateurs souhaitant l’utiliser. De plus, l’utilisation du Datastore nous a fait découvrir comment seront stockées les données d’ici quelques années. C’est-à-dire des grandes tables de données basées sur le NoSQL.