Exemple avec une BD SQLite

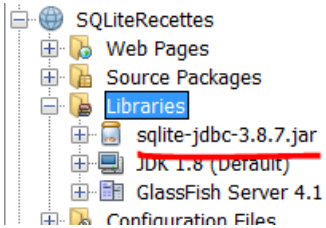
Embarquer une base de données SQLite

Bien souvent, un web-service met à disposition qu'il fournit des données qui lui sont propres à travers les opérations qu'il propose. Rappelons que l'environnement d'exécution est généralement un *Container* (*cf*. précédemment : Tomcat, Geronimo, Websphere, Oracle Application Server, Glassfish).

Dans cette partie, nous mettons en place une base de données embarquées (SQLite) et décrivons les étapes qui permettent de l'exploiter pendant la phase de développement puis de déploiement.

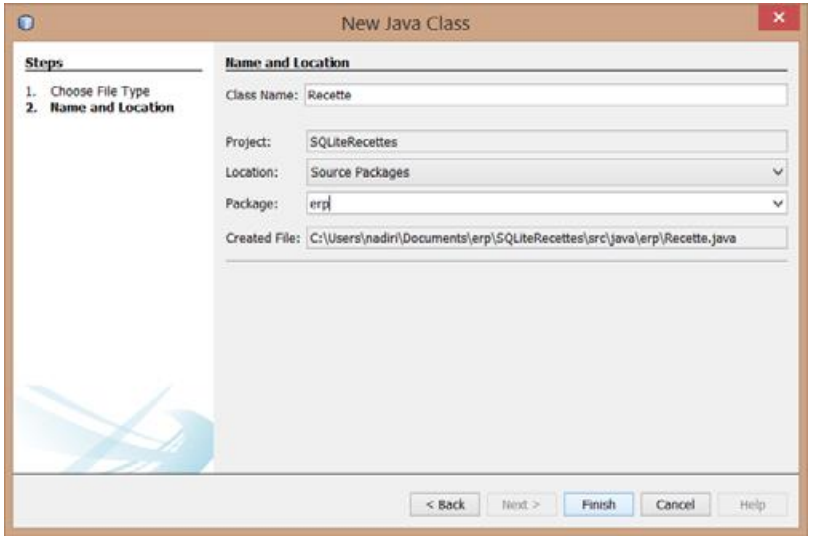
La première étape consiste à inclure la librairie SQLite (à télécharger : https://bitbucket.org/xerial/sqlite-jdbc).

Après avoir créé un projet Web-service nommé SQLiteRecettes, ajouter l'archive jar dans la section *librairies*.



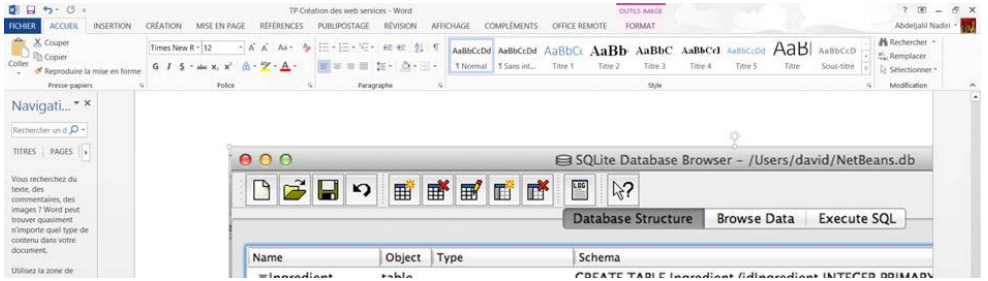
Librairie SQLite dans un projet

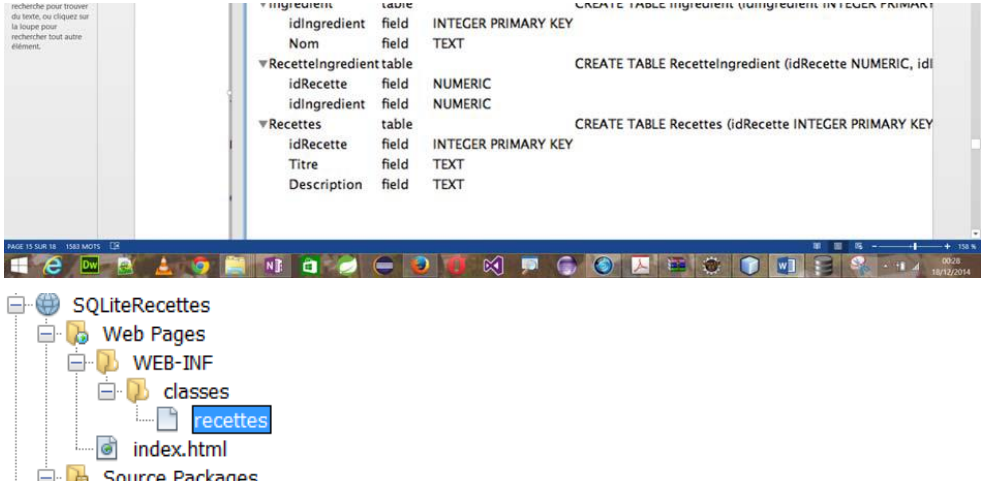
Nous créons ensuite une classe pour tester la librairie et utiliser une base de données test.



Nouveau service de test

A l'aide d'un éditeur de base de données (ici : SQLite DataBase Browser, lien de téléchargement : http://sourceforge.net/projects/sqlitebrowser), nous créons 3 tables permettant de stocker des recettes de cuisine, puis enregistrons la base dans le dossier WEBINF/ classes de notre projet (le dossier classes doit être créé s'il n'existe pas, dans la catégorie other sélectionner folder).





Tables SQlite

Chargement de la base

Nous implémentons ensuite la classe du web-service. Pour que le driver soit chargé au démarrage de la classe, nous plaçons un bloc de code *static* de la manière suivante:

```
package erp;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 *
 * @author nadiri
 */
@WebService(serviceName = "TestSqlite")
public class TestSqlite {

   // Charger le pilote en même temps que la classe
static {
    try{
        Class.forName("org.sqlite.JDBC");
    }
    catch (ClassNotFoundException e)
    {

    }

        }
```

Instanciation du driver JDBC

Nous implémentons ensuite une classe qui va permettre de contenir les données d'une recette dans une structure propre. Il est bien-entendu possible d'utiliser des tableaux, listes et chaînes de caractères comme résultat d'un web service, même si l'on préférera des classes pour des types de données plus complexes.

Le mécanisme qui permet à Java de traduire des classes vers ou à partir des types XML est JAX-B. Lorsque les retours d'opérations utilisent explicitement un type de données complexes, JAX-B se charge automatiquement de faire la traduction.

```java
package erp;

import java.io.Serializable;

/**
 *
 * @author nadiri
 */
public class Recette implements Serializable{
   private String titre;
   private String description;

    public String getTitre() {
        return titre;
    }

    public void setTitre(String titre) {
        this.titre = titre;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }


}
```

Classe Recette : structure de données

Nous implémentons ensuite l'opération qui permet de chercher les recettes qui contiennent un ingrédient donné.

Note

Au moment du déploiement, le web service est embarqué dans un serveur d'applications, qui possède sa propre structure de fichiers. Il est donc fortement déconseillé d'utiliser des chemins absolus pour charger la base de données.

Nous utilisons le *ClassLoader* pour aller charger une ressource, ici notre fichier de base de données. Lors du déploiement, les fichiers, dossiers et jars situés dans le répertoire WEBINF/classes du projet seront copiés tels quels dans le dossier de l'application sur le serveur.

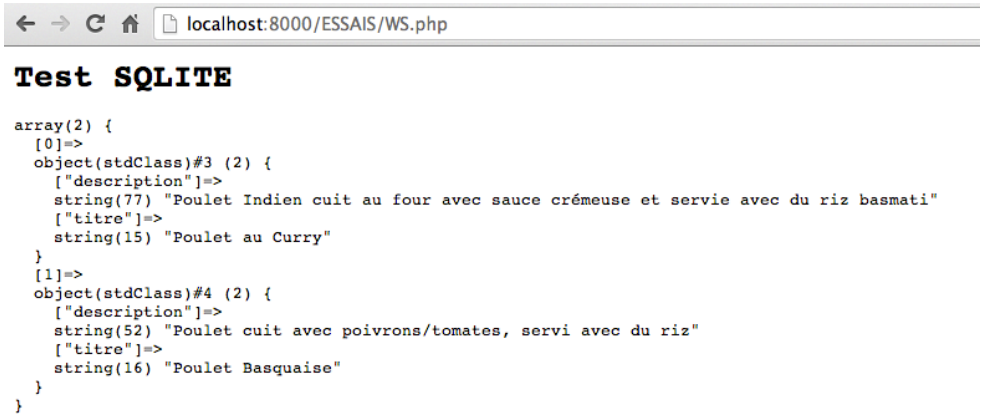
L'opération suivante renvoie une *List* contenant des objets de types recettes.

```java
    */
    @WebMethod(operationName = "rechercheParIngredient")
    public List<Recette> rechercheParIngredient(@WebParam(name = "ingrdient") String ingredient) {
        URL ch = getClass().getResource("/recette");
        String chemin = ch.getPath();
        try {

            Connection cnx = DriverManager.getConnection("jdbc:sqlite:" + chemin);
            Statement st = cnx.createStatement();

            ResultSet rs = st.executeQuery("Select titre, description from recettes"
                    + "inner join RecetteIngredient on RecetteIngredient.idRecette=Recettes.idRecette "
                    + "inner join Ingredient on ingredient.idIngredient=recetteIngredient.idIngrdient "
                    + "where nom like '%" + ingredient + "'"
            );

            List<Recette> resultat = new ArrayList<>();
            while (rs.next()) {
                String titre = rs.getString("Titre");
                String description = rs.getString("Description");
                Recette r = new Recette();
                r.setDescription(description);
                r.setTitre(titre);
                resultat.add(r);
            }
            return resultat;
        } catch (SQLException e) {
            return null;
        }

    }
}
```

Service de recherche de recettes par ingrédient

JAX-B Recette<->XML

Un simple client (ici en PHP) nous renvoie deux résultats, sous la forme d'un classe possédant les deux attributs déclarés dans notre classe recettes.



Consommation du service en PHP