Université de Technologie de Compiègne

Génie Informatique

Rapport de projet - Projet Web services

SR03

Steve LAGACHE & Romain PELLERIN

Chargé de TD : Ahmed Lounis

Printemps 2016 (P16)

Dernière mise à jour : 11 juin 2016

Table des matières

1 XML

1.1 Arbre

Voici un exemple d'arbre XML illustrant les données manipulées pour ce projet.

XML

```xml
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>
<annuaire>
    <annuaire_id>Id</annuaire_id>
    <categorie categorie_id ="Id" name="categorie">
        <annonce>
            <annonce_id>Id</annonce_id>
            <nom>Nom</nom>
            <adresse>
                <adresse_id>Id</adresse_id>
                <rue>Rue</rue>
                <ville>Ville</ville>
                <codepostal>00000</codepostal>
            </adresse>
            <telephone>0000000000</telephone>
        </annonce>
        <annonce>
            <annonce_id>Id</annonce_id>
            <nom>Nom</nom>
            <adresse>
                <adresse_id>Id</adresse_id>
                <rue>Rue</rue>
                <ville>Ville</ville>
                <codepostal>00000</codepostal>
            </adresse>
            <telephone>0000000000</telephone>
        </annonce>
    </categorie>
    <categorie categorie_id ="Id" name="categorie">
        <annonce>
            <annonce_id>Id</annonce_id>
            <nom>Nom</nom>
            <adresse>
                <adresse_id>Id</adresse_id>
                <rue>Rue</rue>
                <ville>Ville</ville>
                <codepostal>00000</codepostal>
            </adresse>
            <telephone>0000000000</telephone>
        </annonce>
    </categorie>
```

```
</annuaire>
```

Et voici sa représentation graphique sous forme d'arbre :

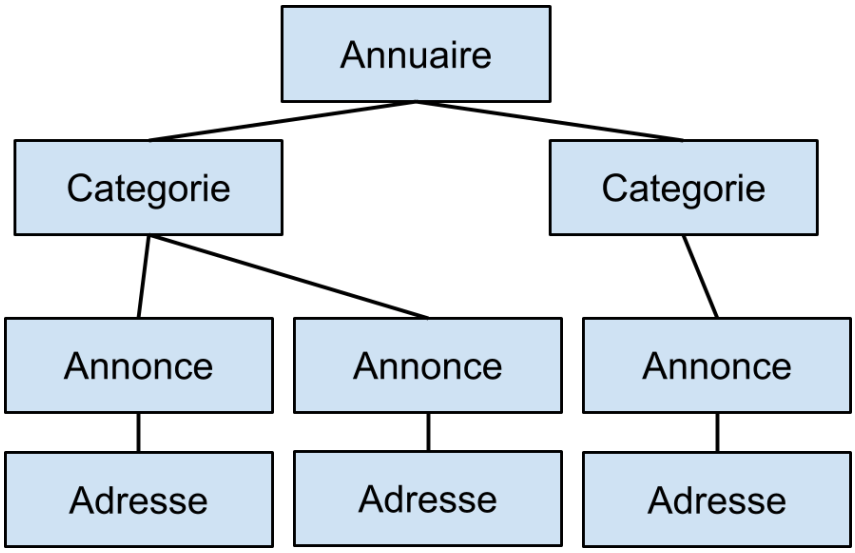

Figure 1.1 – Arbre de l'XML

1.2 Schéma XML

Voici le schéma XSD correspond à notre XML :

XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="
   qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="annuaire">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="annuaire_id"/>
        <xs:element name="categorie" maxOccurs="unbounded"
           minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="annonce" maxOccurs="unbounded"
                 minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="annonce_id"/
                       >
                    <xs:element type="xs:string" name="nom"/>
                    <xs:element name="adresse">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="
                             adresse_id"/>
                          <xs:element type="xs:string" name="rue"/>
```

```
                        <xs:element type="xs:string" name="ville"
                            />
                        <xs:element type="xs:byte" name="
                            codepostal"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element type="xs:byte" name="telephone"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:attribute type="xs:string" name="categorie_id" use=
    "optional"/>
<xs:attribute type="xs:string" name="name" use="
    optional"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

2 Conception

2.1 Diagramme de classes

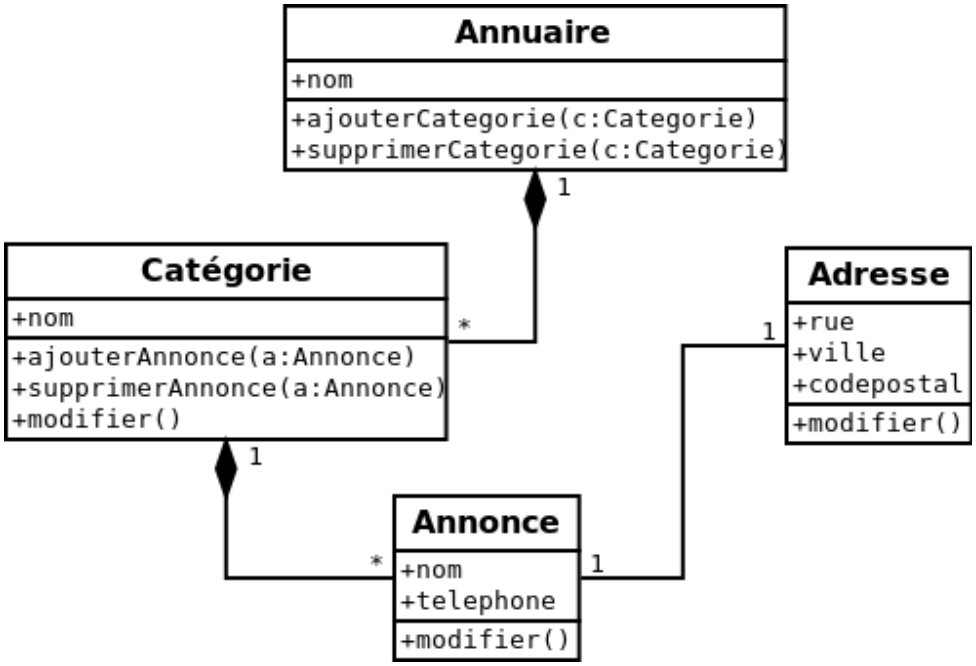

Figure 2.1 – Diagramme de classes (UML)

2.2 Diagramme logique de données (MLD)

XML

```
Annuaire(#nom:string)
Categorie(#nom:string, #annuaire=>Annuaire(nom))
Annonce(#nom:string, #categorie=>Categorie(nom), adresse=>Adresse(
   adresse_id), telephone:string)
Adresse(#adresse_id:integer, rue:string, ville:string, codepostal:
   string)
```

3 Web service

3.1 Interface : prototypes des méthodes

Nous avons décidé de créer deux webservices :

— Un pour l'application cliente d'administration (création, modification et suppression des catégories et annonces)

— Un pour l'application cliente de recherche d'annonces

Voici le webservice d'administration :

Java

```
String createCategory(String name)

String deleteCategorie(int categorie_id)

String updateCategorie(int categorie_id, String name)

String getAllCategorie()

String getCategory(int category_id)

String createAd(String name, int categorie_id, String rue, String
   ville, String codepostal, String telephone)

String deleteAd(int annonce_id)

String updateAd(int annonce_id, String name, int categorie_id,
   String rue, String ville, String codepostal, String telephone)

String getAllAd()

String getAd(int annonce_id)
```

Et voici le webservice pour la recherche d'annonces :

Java

```
String getByParams(String name, String categoryName, String street,
    String city, String postcode)
```

Pour nos deux webservices, chaque fonction retourne une `String` au format JSON.

3.2 Messages SOAP

3.2.1 Requête

Voici une requête faite depuis l'application cliente de recherche d'annonces, vers le serveur :

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
   envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
   xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <getByParams xmlns="http://webservice">
            <name>ad</name>
            <categoryName xsi:nil="true"/>
            <street xsi:nil="true"/>
            <city>paris</city>
            <postcode xsi:nil="true"/>
        </getByParams>
    </soapenv:Body>
</soapenv:Envelope>
```

3.2.2 Réponse

Et voici la réponse à cette requête. Nous retournons du JSON qui est encapsulé dans du XML automatiquement.

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
   envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
   xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <getByParamsResponse xmlns="http://webservice">
            <getByParamsReturn>
                [{&quot;annonce_id&quot;:1,&quot;nom&quot;:&quot;
                   ad11&quot;,&quot;categorie_id&quot;:8,&quot;
                   adresse_id&quot;:1,&quot;telephone&quot;:&quot
                   ;0251&quot;,&quot;categorieObjet&quot;:{&quot;
                   categorie_id&quot;:8,&quot;nom&quot;:&quot;
                   cat2358&quot;},&quot;adresseObjet&quot;:{&quot;
                   adresse_id&quot;:1,&quot;rue&quot;:&quot;rue&
                   quot;,&quot;ville&quot;:&quot;paris&quot;,&quot;
                   codepostal&quot;:&quot;85&quot;}},{&quot;
                   annonce_id&quot;:2,&quot;nom&quot;:&quot;ad2&
                   quot;,&quot;categorie_id&quot;:1,&quot;
                   adresse_id&quot;:2,&quot;telephone&quot;:&quot
                   ;0251&quot;,&quot;categorieObjet&quot;:{&quot;
                   categorie_id&quot;:1,&quot;nom&quot;:&quot;cat1&
                   quot;},&quot;adresseObjet&quot;:{&quot;
                   adresse_id&quot;:2,&quot;rue&quot;:&quot;rue&
                   quot;,&quot;ville&quot;:&quot;parisss&quot;,&
                   quot;codepostal&quot;:&quot;85&quot;}}]
            </getByParamsReturn>
        </getByParamsResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

4 Architecture

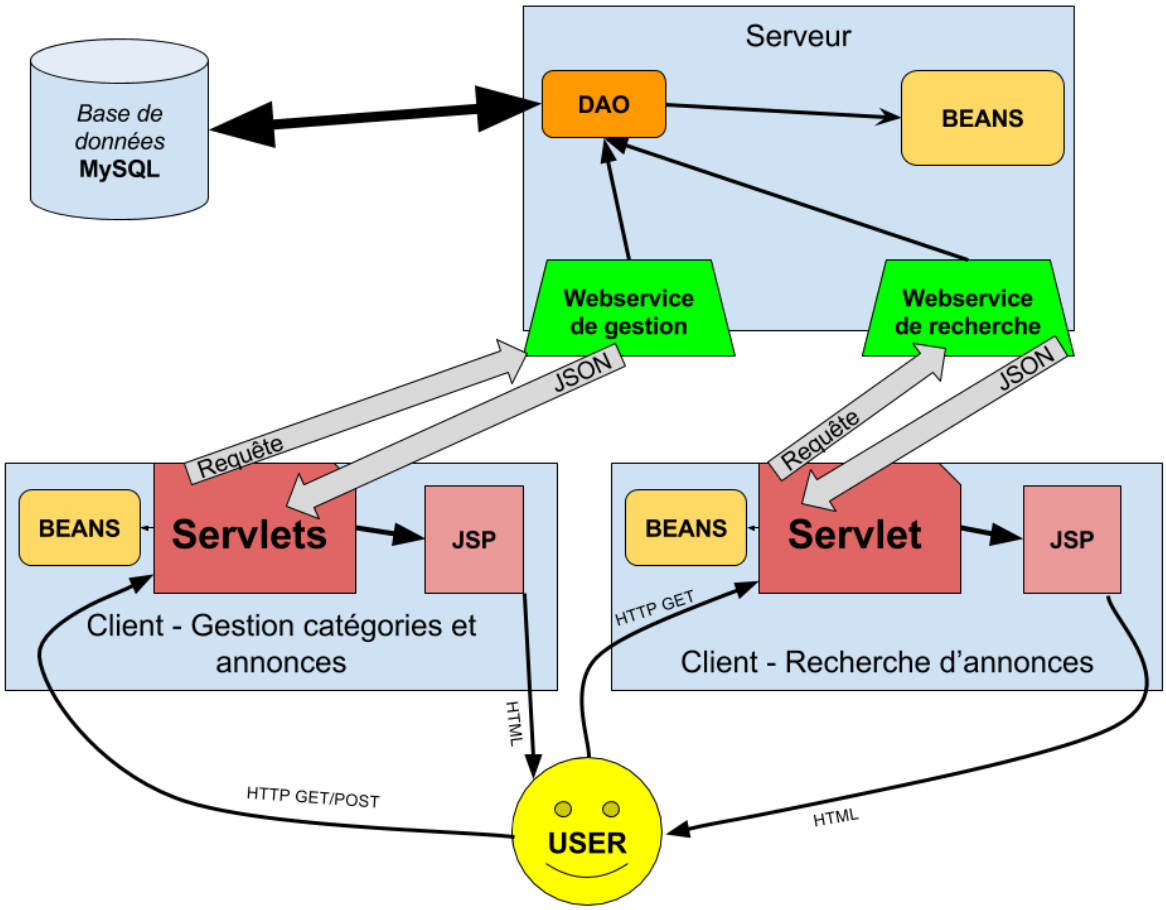

Figure 4.1 – Architecture de l'application

Un utilisateur peut utiliser deux applications web (qui tournent sur Tomcat) au choix :

— Une application web de gestion de l'annuaire, où il peut créer, modifier et supprimer des catégories. Il peut également faire la même chose avec des annonces.

— Une application web de consultation où il peut voir l'ensemble des annonces ou alors faire des recherches sur les annonces pour en trouver seulement certaines, répondant à des critères de recherche.

Lorsqu'il accède à l'une des deux applications, voici ce qu'il se passe :

1. Il envoit une requête HTTP à l'application web pour accèder à une page.
2. Selon les paramètres contenus dans la requête (qui peut être de type GET ou POST), la servlet va soit simplement renvoyer une page HTML générée à partir d'une page JSP, soit faire des traitements plus poussés en utilisant l'API (webservice) de notre serveur, puis renvoyer du HTML généré à partir d'une JSP et de la réponse de l'API.
3. Dans le cas où la servlet a besoin d'utilisateur le web service, elle va utiliser un objet de la classe `Proxy` (en appelant une méthode sur cet objet). De manière transparente, une requête est faite sur le serveur, qui tourne aussi sur Tomcat.

4. La fonction du webservice qui est appelée va instancier une ou plusieurs DAO.
5. Les DAO utilisent les `Beans`, qui sont une représentation des tables SQL que nous avons créées. Les DAO vont donc faire des requêtes SQL et instancier des objets `Beans` à partir des résultats de la requête SQL.
6. La fonction du webservice initialement appelée récupère les résultats produits par la ou les DAO et les sérialise en JSON.
7. Le JSON est retourné à la servlet.
8. La servlet désérialise le JSON et instancie des `Beans`. Ces derniers, souvent sous forme de listes, sont transmis à la JSP qui va produire le HTML renvoyé au client.