Institut Galilée — E-services
Année 2014-2015 — Master 2

T.P. 2

Servlets, beans, base de données

L'objectif de ce TP est de mettre en place un site web pour l'administration et la commande de pizzas :

- une pizza est déterminée par son type et son prix à l'unité
- un stock est donné pour chaque type de pizza
- une commande se définit par un type de pizza, une quantité, le total de la commande, l'email de confirmation

Plus techniquement, l'objectif est de vous faire pratiquer les beans *entity* et *session*, le lien avec une base de données, le lien avec des servlets. Dans ce TP, les tables sont créées manuellement.

1 Base de données

NetBeans contient par défaut la base de données `derby`. Les tables qu'on y déclare se trouvent sous le répertoire `$HOME/.netbeans-derby`.

- L'onglet `Services` donne sous `Databases` les bases de données utilisables (`Java DB` pour `derby`), les drivers disponibles, les connecteurs possibles (un connecteur par base de donnée créée). Un tel connecteur permet d'accéder directement à la base de donnée, de voir les tables et leur contenu, d'effectuer des requêtes SQL soit par l'interface, soit par exécution de la requête explicite directement.
- L'onglet `Services` donne sous `Servers` les serveurs d'application sur lesquels sont déployées les applications développées sous Netbeans. En l'occurence `Glassfish` est fourni en standard. `BEA WebLogic` et `Websphere` sont d'autres serveurs d'application. Ces serveurs intègrent aussi un serveur JNDI qui permettra de faire le lien entre les objets publiés (base de données, servlets, ...).

Pour créer une base de donnée pour le projet : sous l'onglet `Services`, clic droit sur `Java DB - Create Database`

- nom : PizzaDB
- user name : pizza
- password : pizzaUP13
- (port modifiable, 1527 par défaut)

Sous l'onglet `Services`, clic sur le connecteur jdbc de Pizza : `connect`. Deux possibilités alors pour créer des tables : soit via `Create Table`, soit par une requête SQL dans `Execute Command`. La 1ère méthode n'est valable que pour les cas simples (vu l'interface !). Dans les 2 cas, ces options sont disponibles par clic droit sous l'entrée `Tables` du connecteur.

- On commence par créer le schéma PIZZA : Clic droit sur `Tables - Execute Command`,

```
1 CREATE SCHEMA PIZZA;
```

  puis `Execute`
- table Pizza, création des colonnes : Clic droit sur `Tables - Create Table`

  - primary Key, pizza_id, VARCHAR, 10
  - prix, NUMERIC
- table Stock : Clic droit sur `Tables - Execute Command`

```
CREATE TABLE PIZZA."STOCK" (
    "STOCK_ID" INTEGER generated always as identity primary key,
    "PIZZA_ID" VARCHAR(10),
    "QUANTITE" numeric
);
alter table PIZZA.stock add foreign key (pizza_id) references PIZZA.pizza;
```

  puis `Execute`
- table Commande : Clic droit sur `Tables - Execute Command`

```
CREATE TABLE PIZZA."COMMANDE" (
    "COMMANDE_ID" INTEGER generated always as identity primary key,
    "PIZZA_ID" VARCHAR(10),
    "QUANTITE" numeric,
    "TOTAL" numeric,
    "EMAIL" varchar(20)
);
alter table PIZZA.commande add foreign key (pizza_id) references PIZZA.pizza;
```

  puis `Execute`

La commande `refresh` permet de rafraîchir les données d'un schéma. Ajouter par SQL des types de pizza et les stocks associés. Vérifier que la base contient les données.

2 Application et beans

Créer un nouveau projet `Enterprise Application`
- nom : TP_Pizza
- server : GlassFish

2.1 Partie EJB

1. Il faut d'abord ajouter une "unité de persistence", c'est à dire mettre en place un serveur JNDI qui va faire la liaison entre la base de données et les beans : dans `TP_Pizza-ejb, New / Persistence Unit`
   - `Data Source / New Data Source`
   - sélectionner jdbc :... PIZZA
   - mettre PIZZA comme nom JNDI
   - `Table Generation Strategy : None`
2. créer les *Entity Beans* liées aux tables de la BD :
   - `New / Entity Classes from Database`
   - `Data Source : PIZZA`
   - sélectionner les tables
   - mettre `EntityPizza` comme package
   - Finish

   les 3 classes sont créées dans `TP_Pizza-ejb / Source Packages / Pizza`

3. créer les *Session Beans* liées aux *Entity Beans* :
   - `New / Session Beans for Entity Classes`
   - sélectionner les 3 *Entity*
   - créer avec les interfaces Remote et Local
   - mettre `EntityPizza` comme package
   - Finish
4. Modifier `StockFacade` en ajoutant :

```
1     public void create(String pizzaId, int quantite) {
2         String s = "insert into stock(pizza_id,quantite) ";
3         s = s + "values ('" + pizzaId + "'," + quantite + ")";
4         em.createNativeQuery(s).executeUpdate();
5     }
```

   et `StockFacadeLocal` en ajoutant :

```
1     void create(String pizzaId, int quantite);
```

5. Modifier `CommandeFacade` en ajoutant :

```
1     public void create(String pizzaId, int quantite, int total, String email) {
2         String s = "insert into commande(pizza_id,quantite,total,email) ";
3         s = s + "values ('" + pizzaId + "'," + quantite + ",";
4         s = s + total + ",'" + email + "')";
5         em.createNativeQuery(s).executeUpdate();
6     }
```

   et `CommandeFacadeLocal` en ajoutant :

```
1     void create(String pizzaId, int quantite, int total, String email);
```

2.2 Partie servlets

Dans `TP_Pizza-war`, créer les servlets d'administration et de gestion de commandes par `New / Servlet` pour

- `Admin_Pizza` dans le package `Administration` (faire `Next` pour définir le nom public)
  - ajouter le code "noyau" (cf annexe A) à la place de ce qui est en commentaire entre `try` et `finally`
  - sélectionner `clic droit / Insert Code ... / Call Enterprise Bean` (ou pour une version antérieure de Netbeans `pizzaFacade / clic droit / Enterprise Resources / Call Enterprise Bean`)
  - sélectionner `pizzaFacade` de `TP_Pizza-ejb`
  - ajouter enfin les import nécessaires
- `Commande_Pizza` dans le package `GestionClient`
  - ajouter le code "noyau" (cf annexe B) à la place de ce qui est en commentaire entre `try` et `finally`
  - sélectionner `clic droit / Insert Code ... / Call Enterprise Bean` (ou pour une version antérieure de Netbeans `stockFacade / clic droit / Enterprise Resources / Call Enterprise Bean`)
  - sélectionner `stockFacade` de `TP_Pizza-ejb`
  - ajouter enfin les import nécessaires

2.3 Utilisation

Faire `Build` sur les 2 sous-projets `TP_Pizza-ejb` et `TP_Pizza-war`. Dans `TP_Pizza` : `undeploy / deploy`

Requêtes http (le numéro de port est mentionné dans la fenêtre `GlassFish`) :

- pour l'administration
  `http://localhost:8093/TP_Pizza-war/Admin_Pizza`
- pour la commande
  `http://localhost:8093/TP_Pizza-war/Commande_Pizza`

3 Développements

Modifier et compléter le code pour les fonctionnalités suivantes :

- Mettre à jour le stock après chaque commande
- Refuser les commandes pour un stock insuffisant, refuser les commandes pour un type inexistant
- Lister les commandes effectuées
- Envoyer un mail une fois la commande effectuée

A Code noyau pour `Admin_Pizza`

```
1               out.println("<html>");
2               out.println("<head>");
3               out.println("<title>Administration Pizza</title>");
4               out.println("</head>");
5               out.println("<body>");
6               out.println("<h1>Ajouter un nouveau type de pizza : </h1>");
7               List lPizza = pizzaFacade.findAll();
8               for (Iterator it = lPizza.iterator(); it.hasNext();) {
9                   Pizza elem = (Pizza) it.next();
10                  out.println("Type : <b>" + elem.getPizzaId() + " </b> ");
11                  out.println("Prix : " + elem.getPrix() + "<br/>");
12              }
13              String type=null;
14              type=request.getParameter("type");
15              if (type!=null) {
16                  try {
17                      int prix=0;
18                      prix=new Integer(request.getParameter("prix"));
19                      int quantite=new Integer(request.getParameter("quantite"));
20                      out.println("On ajoute un type de pizza<br/>");
21
22                      Pizza e = new Pizza();
23                      e.setPizzaId(type);
24                      e.setPrix(prix);
25                      pizzaFacade.create(e);
26                      stockFacade.create(type,quantite);
27                      response.sendRedirect("Admin_Pizza");
28
29                  } catch (Exception ex) {
30                      ex.printStackTrace();
```

```
31                  }
32              out.println("Pizza sauvegardé");
33              } else {
34                  out.println("<form method='POST'>");
35                  out.println("Type: <input type='text' name='type'><br/>");
36                  out.println("Prix: <input type='text' name='prix'><br/>");
37                  out.println("Quantité: <input type='text' name='quantite'><br/>");
38                  out.println("<input type='submit'><br/>");
39                  out.println("</form>");
40              }
41              out.println("</body>");
42              out.println("</html>");
```

B Code noyau pour `Commande_Pizza`

```
1               out.println("<html>");
2               out.println("<head>");
3               out.println("<title>Commande Pizza</title>");
4               out.println("</head>");
5               out.println("<body>");
6               List lStock = stockFacade.findAll();
7               for (Iterator it = lStock.iterator(); it.hasNext();) {
8                   Pizza elem = ((Stock) it.next()).getPizzaId();
9                   out.println("Type : <b>" + elem.getPizzaId() + " </b> ");
10                  out.println("Prix : " + elem.getPrix() + "<br/>");
11              }
12
13              out.println("<h1>Choisissez votre pizza : </h1>");
14              String type=request.getParameter("type");
15              if (type!=null) {
16                  try {
17                      int quantite=new Integer(request.getParameter("quantite"));
18                      String email=new String(request.getParameter("email"));
19                      Pizza pizza = pizzaFacade.find(type);
20                      int total = quantite * pizza.getPrix();
21                      out.println("On ajoute une commande dont le prix total est :"+tot
22
23                      commandeFacade.create(type, quantite, total, email);
24
25                  } catch (Exception ex) {
26                      ex.printStackTrace();
27                  }
28              out.println("Commande effectuée");
29              } else {
30                  out.println("<form method='POST'>");
31                  out.println("Type: <input type='text' name='type'><br/>");
32                  out.println("Quantité: <input type='text' name='quantite'><br/>");
33                  out.println("Email: <input type='text' name='email'><br/>");
34                  out.println("<input type='submit'><br/>");
35                  out.println("</form>");
36              }
37              out.println("</body>");
38              out.println("</html>");
```