JSF2 et EJB3 Session

Ce document est composé de 2 tutoriaux.

Le premier montre comment concevoir et développer le use case ‘Créer son profil’ avec JSF. A ce stade l'objectif est de comprendre JSF dans ce qu'il y a de commun à toute version, que ce soit la version 1.2 ou la version 2.

Une seconde partie aborde les apports de la version 2 de JSF

Le second tutorial expose les EJB3 session à partir de l'application Petstore et explicite l'injection de dépendance. On montre notamment comment implémenter un EJB3 stateful avec JSF.

La fin du document définit les étapes pour configurer l'application Petstore sous Jboss-5.1.0.GA.

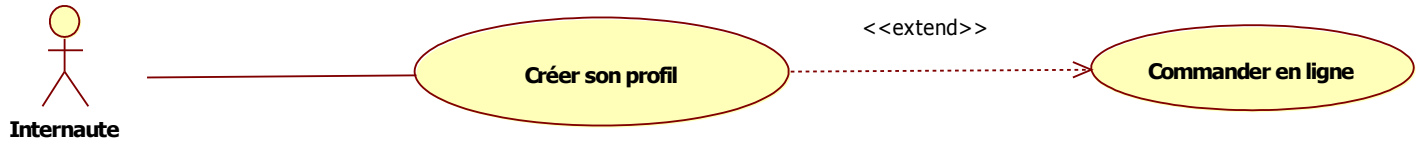
J. HILDEBRAND

Première partie : Petstore en JSF2

I. Les bases de JSF à partir du use case 'Créer son profil'

1. Expression des besoins

Les internautes doivent saisir leurs informations : login, identité, adresse et carte bancaire pour être connus du système et passer leurs commandes en ligne.

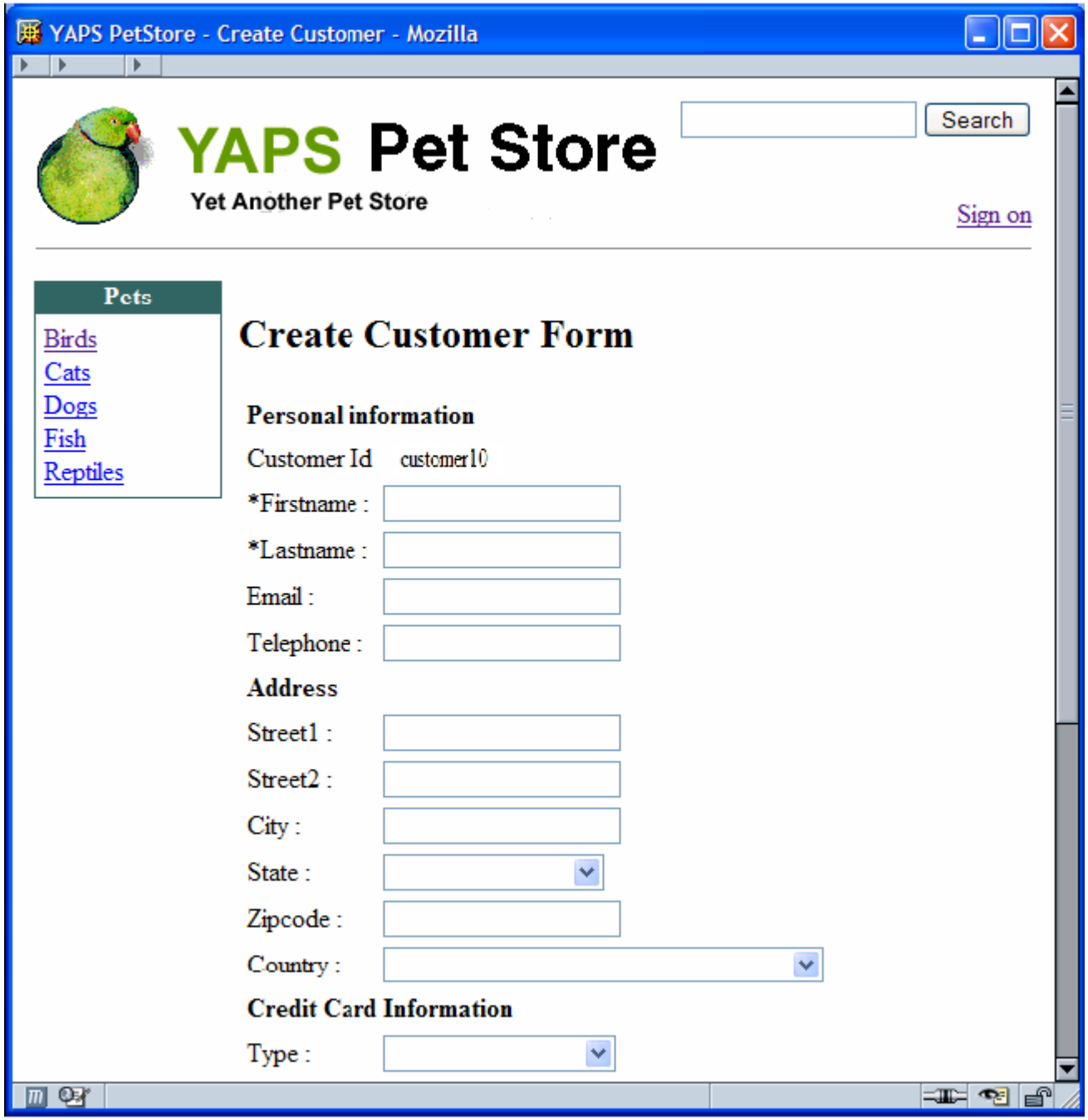


1.1 Use case Créer son profil

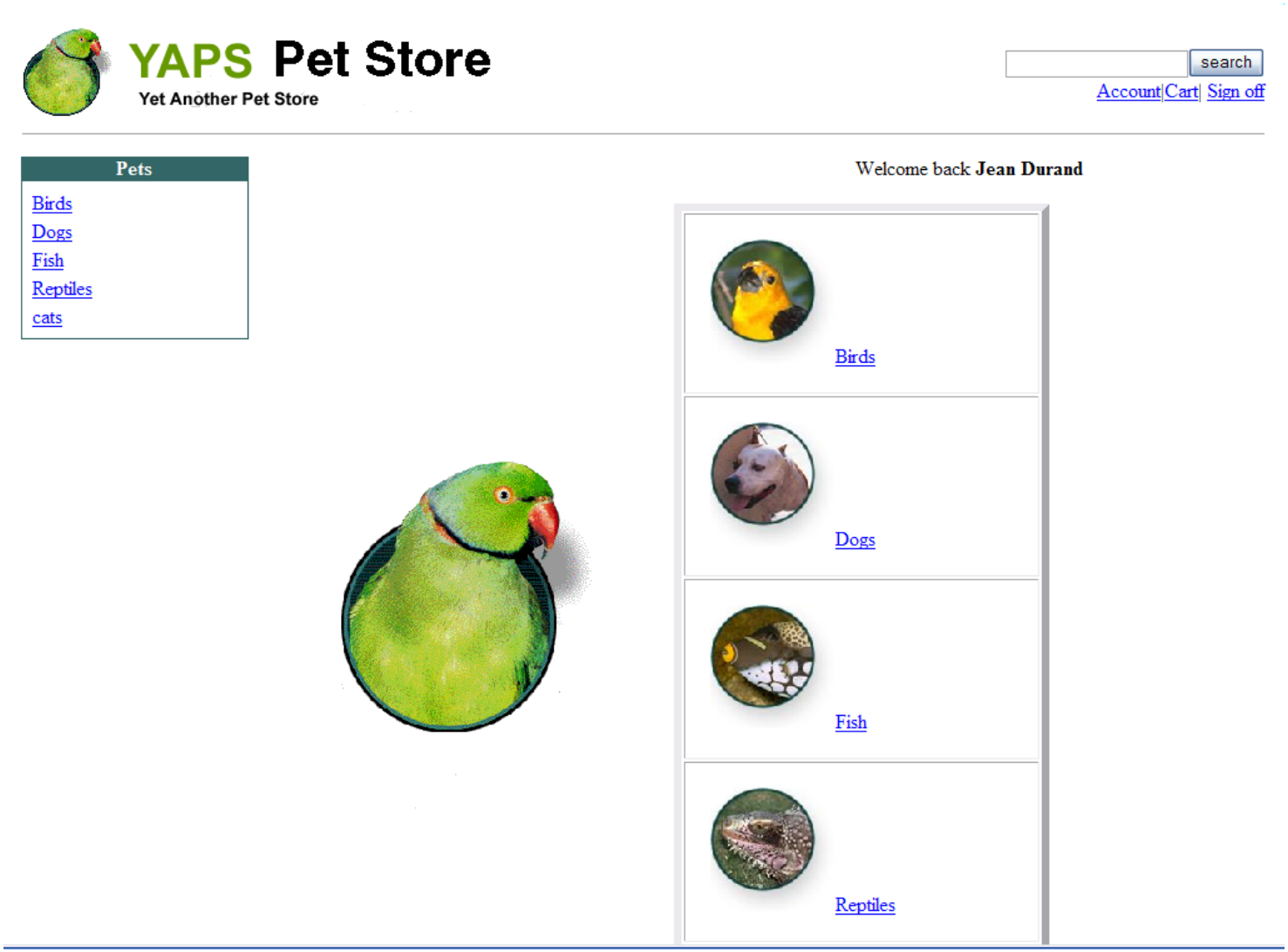
Nom	Créer son profil
Résumé	Permet à un internaute de créer son login puis de saisir ses informations personnelles pour être connu du système et pouvoir passer commande sur le catalogue Petstore.
Acteur	Internaute
Pré-conditions	
Description	**1.** L'internaute rempli les **informations de login** : Customer id (*) Password (*) Password (Repeat) (*) **2.** Dès qu'il valide ces données, le système contrôle la conformité de la saisie, puis redirige le user sur le formulaire des **informations personnelles** : First Name (*) Last Name (*) Telephone Email Street 1 Street 2 City State Zipcode Country Credit Card Number Credit Card Type

	Credit Card Expiry Date : le format de cette date est MM/AA. Les champs (*) sont obligatoires.
Exceptions	[1] Une exception de validation `CheckException` est levée si l'un des champs obligatoires est manquant. [2] Si le customer id existe déjà dans le système une exception `createException` est levée. [3] Si les deux mots de passe ne sont pas identiques, une exception `CheckException` est levée. [4] Si le format de la date d'expériation de la carte de crédit n'est pas respecté une exception `CheckException` est levée.
Post-conditions	Le profil de l'internaute est créé. Il est logé au système et peut remplir son cady en consultant le catalogue : – la page du catalogue est personnalisée en mentionnant le prénom et le nom de l'internaute. – les liens 'Account' (pour mise à jour du compte par l'internaute) et 'Cart' (pour visualiser le contenu du cady) sont accessibles sur la page.

1.2 Ecrans du use case


YAPS PetStore - Create Customer - Mozilla
YAPS Pet Store
Yet Another Pet Store
Search
Sign on
Pets
Birds
Cats
Dogs
Fish
Reptiles
Create Customer Form
Personal information
Customer Id customer10
*Firstname :
*Lastname :
Email :
Telephone :
Address
Street1 :
Street2 :
City :
State :
Zipcode :
Country :
Credit Card Information
Type :

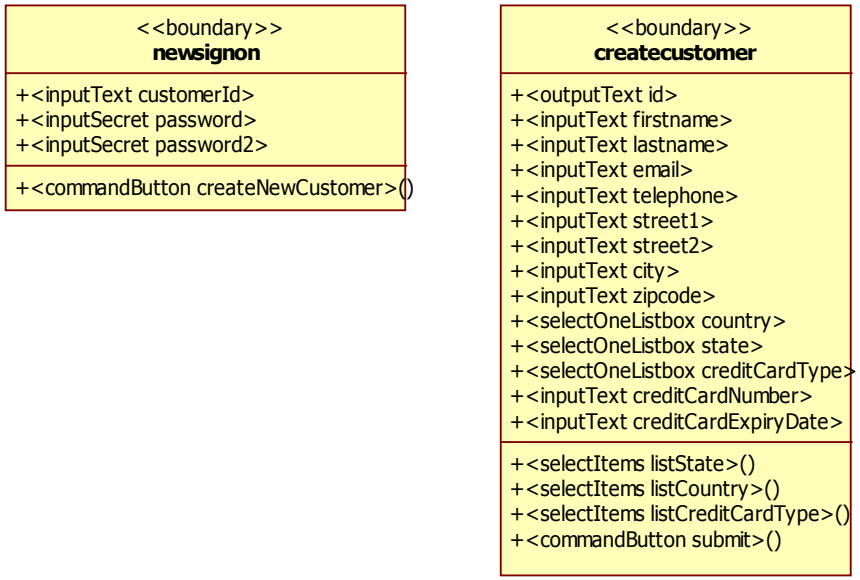
Post-condition : redirection sur le catalogue



2. Conception

Modèle objets

Les écrans sont constitués de composants graphiques JSF.

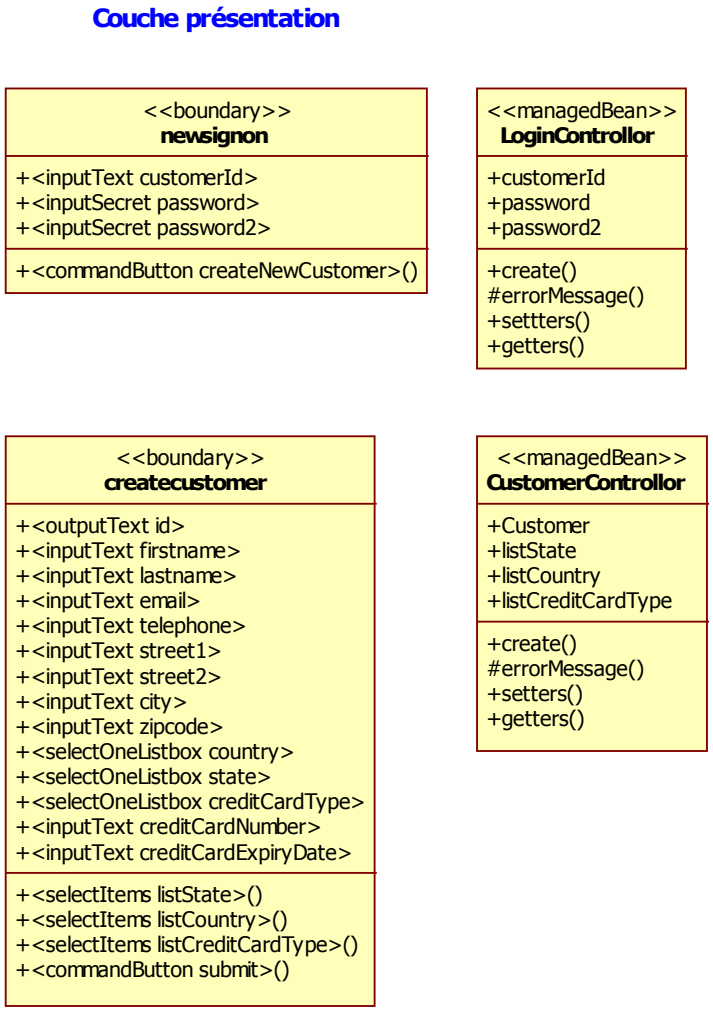


Seuls sont représentés les composants qui participent à la création du profil par le système : mise en évidence de l'essentiel du système à modéliser.

No. I would like to sign up for an account.

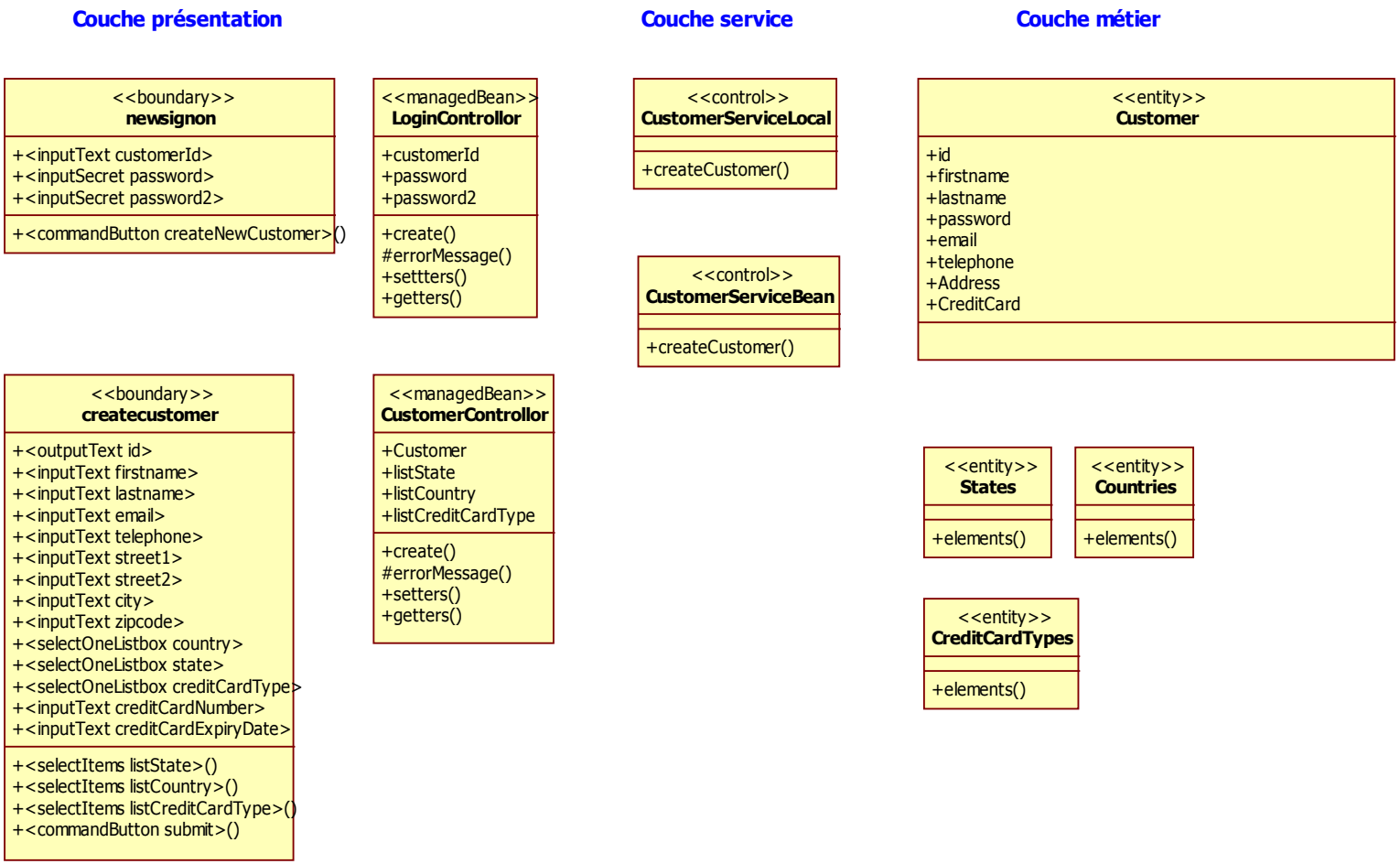

```
<h:panelGrid columns="2>
    <h:outputText value="Customer Id:"/>
    <h:inputText id="idCustomer1" size="15" value="#{login.customerId1}"/>
    <h:outputText value="Password:"/>
    <h:inputText id="idPassword1" size="15" value="#{login.password1}"/>
    <h:outputText value="Password (Repeat):"/>
    <h:inputText id="idPassword2" size="15" value="#{login.password2}"/>
</h:panelGrid>
<h:commandButton id="createNewAccount" value="Create New Account" action="#{login.create}"/>
```

Les valeurs des champs ou les attributs 'action' sont mappés sur les beans gérés par JSF.



Indépendance couche présentation et couche métier : une modification de l'IHM (utilisation d'un commandLink au lieu d'un commandButton) n'induit pas une modification des objets métier.

Les composants graphiques sont reliés au système à un instant donné.



3. La page Jsf

En JSF2 une page suit le squelette ci-après :

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html" >

    <f:view>
        <h:form id="createcustomerForm">
            ...
        </h:form>
    </f:view>
</html>
```

xhtml est le langage par défaut de la version 2 de JSF. Il combine le xml et le html 4.
Core est une bibliothèque de JSF de composants et fonctionnalités. Elle ne gère pas le rendu.
Html est une bibliothèque de composants graphiques avec le rendu html.

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:h="http://java.sun.com/jsf/html" >

    <f:view>
        <h:form id="createcustomerForm">

       <h:outputText value="Lastname* :"/>
       <h:inputText value="#{customercontrollor.customer.lastname}"/>
        ...
       <h:outputText value="State :" />
       <h:selectOneListbox id="state" value="#{customercontrollor.customer.state}" >
           <f:selectItems value="#{customercontrollor.listState}" />
       </h:selectOneListbox>
        ...
       <h:commandButton value="Submit"
                        action="#{customercontrollor.create}"/>

        </h:form>
    </f:view>
</html>
```

4. Le bean managé

Un bean managé a au minimum un constructeur sans argument et des accesseurs. Il est associé (bindé) à une ou plusieurs pages xhtml.

Les attributs du bean sont mappés sur les valeurs à saisir de la page :

```
private CustomerDTO customerDTO;

private SelectItem[] listState ;
private SelectItem[] listCountry ;
private SelectItem[] listCreditCardType ;

@EJB (name ="petstore/CustomerServiceBean/remote")
CustomerServiceRemote customerBean;
```

Les accesseurs permettent la mise à jour des attributs :

```
// UPDATE MODELE
public CustomerDTO getCustomerDTO() {
     return customerDTO;
}

public void setCustomerDTO(CustomerDTO customerDTO) {
     this.customerDTO = customerDTO;
}
....
```

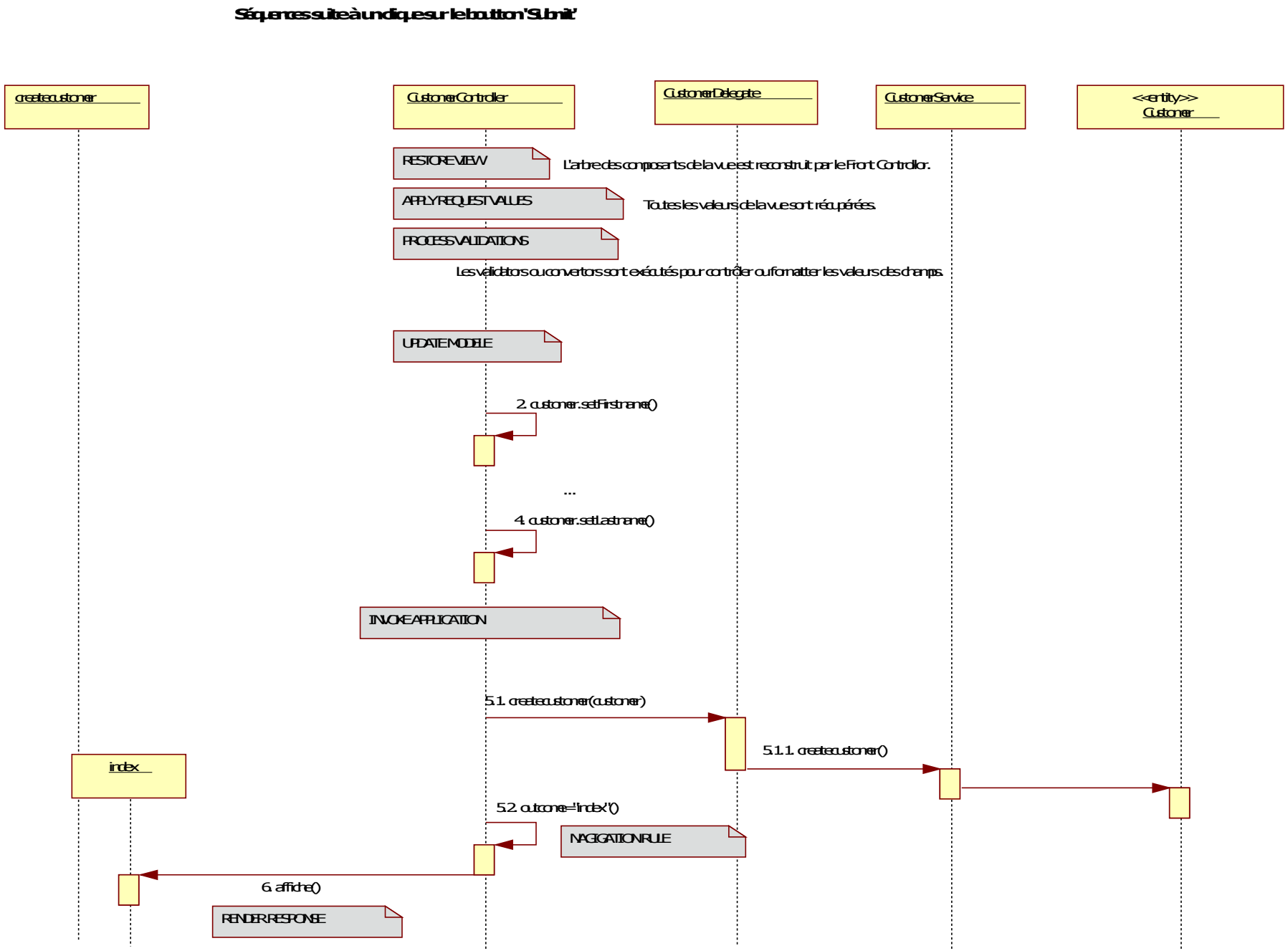
La méthode create correspond à l'attribut action du commandButton de la page

```
// INVOKE APPLICATION
public String create(){
     String response =null;
     final String mname = "create";
     Trace.entering(this.getClass().getSimpleName(), mname);

     try {
       // Creates the customer
       customerDTO =customerBean.createCustomer(customerDTO);

       // NAVIGATION RULE - RENDER RESPONSE
       response ="index";
     }
     catch (Exception e) {
       // RENDER RESPONSE
       if (e.getCause() instanceof  DuplicateKeyException) {
     errorMessage(e.getCause().getClass().getSimpleName()                    + " : Sign On again.
Id already exists.");
       }
       else {
                    errorMessage(e.getMessage());
       }
      }
      return response;
     }
```

5. Diagramme de séquences - Le code du bean suit le cycle de vit d'une requête JSF.



6. Le fichier faces-config.xml

Déclare les beans managés par JSF et définit les règles de navigation.
A l'instanciation de CustomerControllor, la propriété customerDTO est valuée dès l'ouverture de la page. Elle est issue du login.

```
<managed-bean>
  <description>login</description>
  <managed-bean-name>login</managed-bean-name>
  <managed-bean-class>
         com.yaps.petstore.web.jsf.LoginControllor
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
 <description>customercontrollor</description>
 <managed-bean-name>customercontrollor</managed-bean-name>
 <managed-bean-class>
         com.yaps.petstore.web.jsf.CustomerControllor
 </managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
 <property-name>customerDTO</property-name>
    <value>#{login.customerDTO}</value>
 </managed-property>
</managed-bean>
```

II. Les apports de JSF2

Adoption des **facelets** comme **standard de JSF2**.

Avec les facelets, **JSF2** place au coeur de son framework **la logique objet** dans la conception des interfaces graphiques.

Les **facelets** sont tout à la fois un modèle de conception et une technologie.

1. Motivations

- Permettre la **séparation des métiers** entre le designer et le développeur JSF :
  les designers peuvent visualiser les pages en dehors d'un serveur Tomcat.

  Exemple de la page createcustomer_view_designers.xhtml

- Fragmenter **les vues** pour en faire **des composants réutilisables**. C'est le design pattern **composite-view**. Il est associé à la technique du **templating**.

- **Des logiciels dédiés aux designers**. Permettent de travailler avec **Dreamweaver** et de visualiser les composants JSF en dehors de tout serveur d'application. Utilisent des designs pattern qui sont des modèles objets graphiques avec pour objectifs la réutilisabilité pour faciliter les évolutions.

2. Les facelets

Deux aspects

un framework qui s'appuient sur le design pattern composite-view
une technologie qui compile les tags des vues de façon plus efficace

2.1 Un **framework** qui s'appuient sur le **design pattern composite-view**

Une vue composite est une vue utilisant des fragments de vues réutilisables. Un changement sur un fragment de vue est automatiquement répercuté sur chaque vue composite qui l'utilise. De plus, une vue composite contrôle le look and feel de ses vues subalternes à partir du template. Le template centralise tout le look and feel de l'application. (Sun java Blueprints).

Une facelet est une section de page réutilisable ou un ensemble de composants dans l'arbre JSF qui peut elle même être constituée d'autres facelets et/ou inversement être utilisée par une facelet parent.

Pour implémenter le design-patern composite-view, **différents tags** sont proposés : le composant, la composition, la décoration ...

Les vues jsf de Petstore sont organisées en 3 répertoires :

composition
template
facelets

Une vue est une composition (répertoire composition) de fragments de pages (répertoire facelets). Le look and feel de l'ensemble est assuré à partir d'un template.

2.2 Une **technologie** qui **compile les tags** des vues **de façon plus efficace**.

Les facelets **remplacent le standard JSP** dans JSF2.

Dans un premier temps, une facelet utilise un **compilateur XHTML**. A ce stage xhtml vérifie la grammaire des balises.

Dans un second temps, une facelet **utilise SAX** pour compiler les différents éléments de la vue. Elle construit dans un objet Facelet l'arbre des tags de la vue.
Le prosessus de compilation des facelets est plus rapide que celui des jsp, car il n'y a pas de byte code java à générer et compiler lors de toute initialisation de page.

Seconde partie : Petstore en EJB3 session

Le tutorial expose les EJB3 session à partir de l'application Petstore et explicite l'injection de dépendance. L'architecture en couches est mise en évidence :

Les **EJB3 session** (couche service) utilisent une data source configurée dans Jboss pour autoriser une fonctionnalité des EJB : la gestion des transactions par le container.

Les **DAO** (couche métier) ne gèrent pas les transactions : cette fonctionnalité est déléguée au niveau (supérieur) des méthodes des services de l'EJB pour déclencher un rollback général, lors de la survenue d'une erreur, dans une série de mises-à-jour et/ou contrôles.

Les **beans** managés de **JSF** (couche cliente) utilisent l'injection de dépendance pour accéder aux EJB session.

I. EJB stateless

1. Sur **le bean** (couche service) CustomerServiceBean

L'annotation **@Stateless** indique au conteneur qu'il s'agit d'un ejb stateless. Le bean n'a plus à implémenter les méthodes EJB Callback spécifier dans l'interface javax.ejb.SessionBean.

La gestion des transactions est prise en charge par le conteneur pour utiliser les fonctionnalités des EJB3.

```
@TransactionManagement(value=TransactionManagementType.CONTAINER)
```

A condition de définir **une data source** gérée par le conteneur.

```
@Resource(name="PetstoreDS", mappedName="java:PetstoreDS")
   private DataSource ds;
```

La politique transactionnelle se définit **au niveau des méthodes de la couche service**. Chaque méthode délimite la liste des actions à réaliser au sein d'une transaction.

```
@TransactionAttribute(value = TransactionAttributeType.REQUIRED)
   public CustomerDTO createCustomer(final CustomerDTO customerDTO) throws CreateException, CheckException {...}
```

2. Sur **l'interface distante** CustomerServiceRemote

L'annotation @Remote indique au conteneur qu'il s'agit de l'interface. Les méthodes n'ont plus à spécifier 'throw RemoteException' issu de l'interface javax.ejb.EJBObject.

Les fichiers de configuration de l'ejb sont à supprimer.

II. EJB stateful

1. Expression des besoins

L'internaute rempli son caddy d'articles en consultant le catalogue.

Dès qu'il appuie sur le lien Check Out, la commande est validée.

Les fichiers de déploiement de l'ejb stateful sont à supprimer.

2. Injection de dépendance

La classe **ShoppingCartLocator** a pour objet de récupérer une référence distante (`ShoppingCart`) sur l'ejb stateful (`ShoppingCartBean`). En EJB3, un lookup sur le contexte est suffisant : la méthode 'create' de l'EJBHome n'est pas invoquée.

Cette classe est déclarée dans le fichier faces-config pour être **managée par JSF avec une portée session.** L'ejb stateful est injecté (instancié) par JSF à chaque fois qu'un utilisateur ouvre un navigateur. Les choix de l'internaute sont mémorisés pendant la durée de vie de la session. 2 utilisateurs ouvrant 2 navigateurs (et 2 sessions distinctes) auront des caddy distincts.

Classe ShoppingCartLocator

```
/** Locator de l'ejb stateful. Bean managé par JSF de portée session */
public class ShoppingCartLocator {

    /** Propriété managé.
     * Référence de l'ejb stateful.
     * La valeur est injectée par JSF
     *
     */
private ShoppingCart shoppingCardUser=getShoppingCart();


public ShoppingCart getShoppingCardUser() {
    return shoppingCardUser;
```

```
}

public void setShoppingCardUser(ShoppingCart shoppingCardUser) {
     this.shoppingCardUser = shoppingCardUser;
}


public static ShoppingCart getShoppingCart() {
     final String mname = "getShoppingCart";
     ShoppingCart shoppingCartRemote =null;
     try {
          // Looks up for the remote interface
          Context context = new InitialContext();
           shoppingCartRemote =
(ShoppingCart)context.lookup("petstore/ShoppingCartBean/remote");
     }
     catch (Exception e) {
                    Trace.throwing("ShoppingCartDelegate", mname, e);
                }
           return shoppingCartRemote;

     }

}
```

Fichier faces-config

```
<managed-bean>
 <description>shoppingCard</description>
 <managed-bean-name>shoppingCard</managed-bean-name>
 <managed-bean-class>          com.yaps.petstore.common.locator.ShoppingCartLocator
 </managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
</managed-bean>

<managed-bean>
 <description>caddy</description>
 <managed-bean-name>caddy</managed-bean-name>
 <managed-bean-class>
    com.yaps.petstore.web.jsf.CaddyControllor
 </managed-bean-class>
 <managed-bean-scope>session</managed-bean-scope>
 <managed-property>
     <property-name>customerDTO</property-name>
      <value>#{login.customerDTO}</value>
  </managed-property>
  <managed-property>
     <property-name>shoppingCardUser</property-name>
     <value>#{shoppingCard.shoppingCardUser}</value>
  </managed-property>
</managed-bean>
```

3. Client JSF - Classe CaddyControllor

Le bean managé `CaddyControllor` (associé à la page cart.jsp) récupère l'ejb (shoppingCardUser), initialement instancié par JSF. Il n'est plus nécessaire dans le code d'utiliser `ShoppingCartLocator.getShoppingCart()` pour créé l'ejb.

```
/**
 * Bean JSF associé à la page cart.jsp : affiche et met à jour le caddy du user.
 */
public class CaddyControllor extends Controllor {

    ...

 // Référence sur l'ejb stateful ShoppingCardBean
 ShoppingCart shoppingCardUser ;


 public String remove() {
        ...
    shoppingCardUser.removeItem(itemId);
    // Gets the content of the Shopping Cart and
    cartItemsDTO = shoppingCardUser.getItems();
    // ... the total of the shopping cart and
    total = shoppingCardUser.getTotal();

        ...
 }
```

Installation sous Jboss 5.1.0.GA

1 Déclarer les librairies sous eclipse

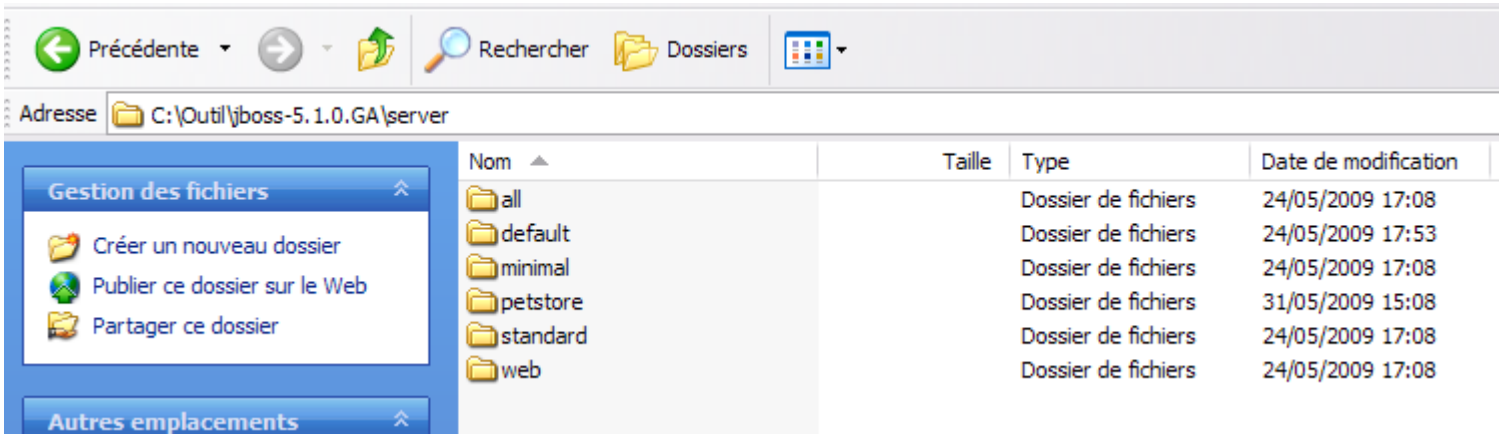
Ouvrir le projet sous eclipse :
Projet, Java Build Path, Add External JARs

Ajouter la liste de librairies.
Les jars se trouvent sous le répertoire lib du projet.

Pour faire tourner les tests Junit sous eclipse, ajouter dans les librairies du projet les fichiers jars du répertoire C:\Outil\jboss-5.1.0.GA\client.

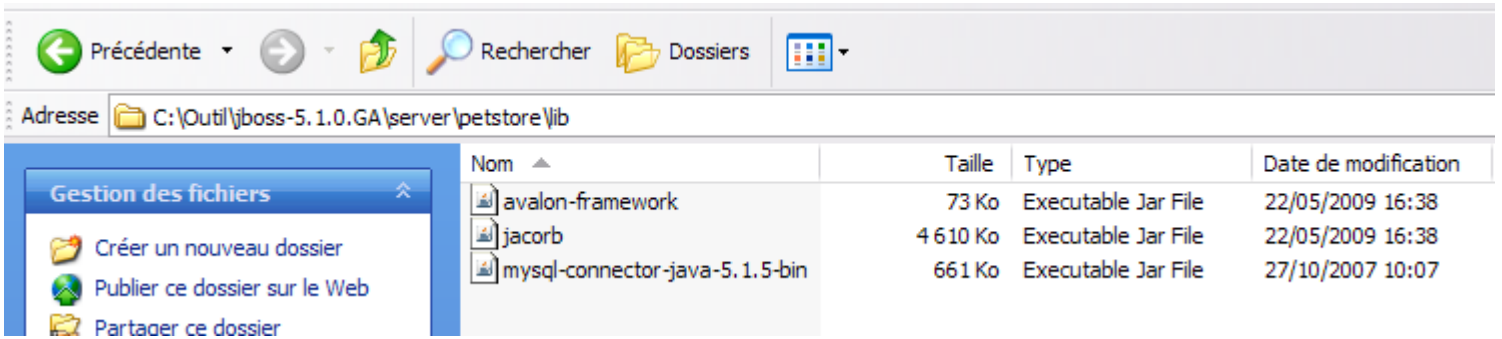
2 Créer le répertoire petstore sous JBoss

à partir d'une copie du répertoire standard.

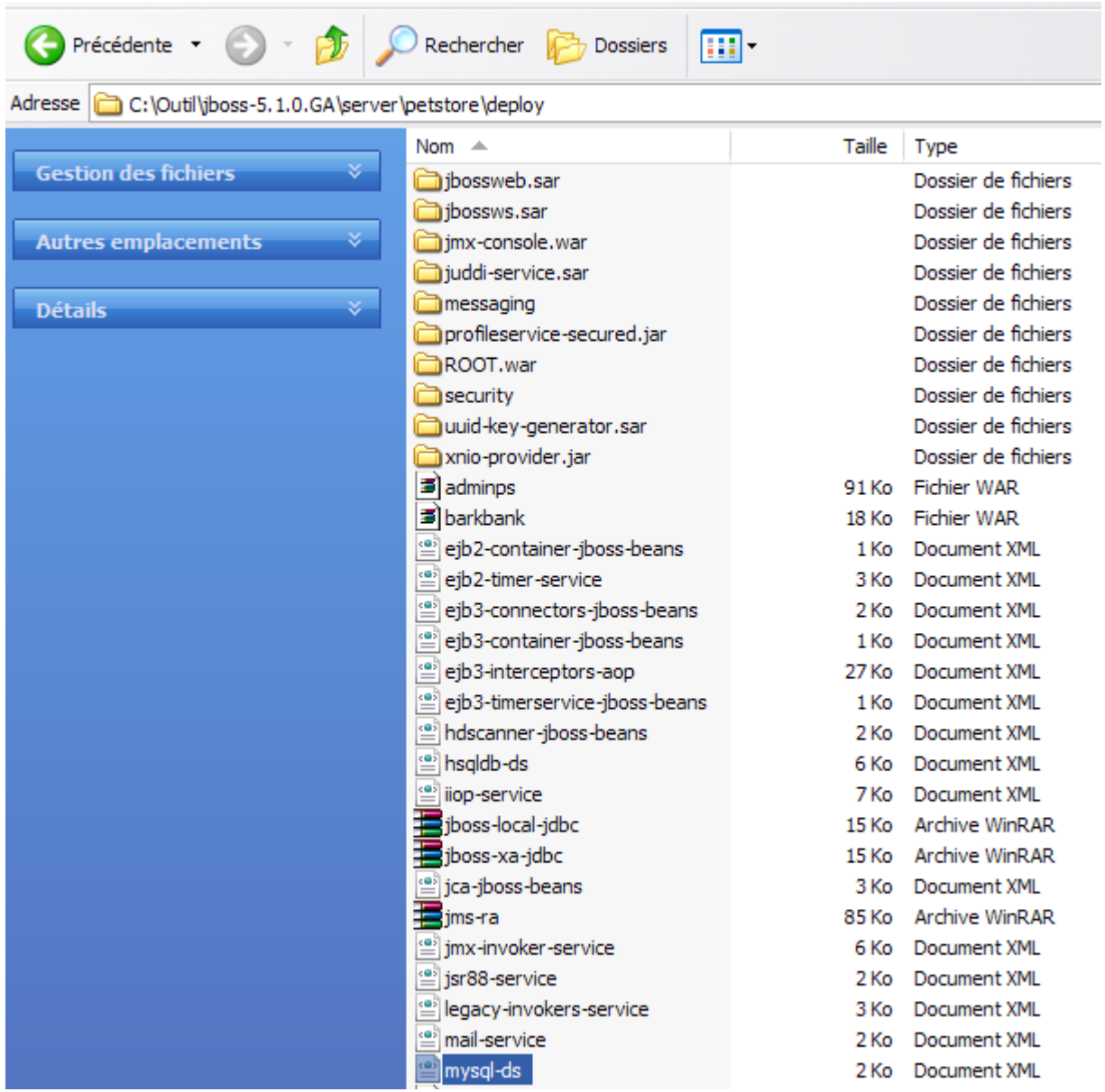


3 Déclarer les librairies sous Jboss

Ajouter la **librairie de mysql** dans le répertoire lib de petstore :



4. Copier dans le répertoire deploy de petstore le fichier mysql-ds.xml qui définit la data source :


Précédente
Rechercher
Dossiers
Adresse
C:\Outil\jboss-5.1.0.GA\server\petstore\deploy
Gestion des fichiers
Autres emplacements
Détails
Nom
Taille
Type
jbossweb.sar
Dossier de fichiers
jbossws.sar
Dossier de fichiers
jmx-console.war
Dossier de fichiers
juddi-service.sar
Dossier de fichiers
messaging
Dossier de fichiers
profileservice-secured.jar
Dossier de fichiers
ROOT.war
Dossier de fichiers
security
Dossier de fichiers
uuid-key-generator.sar
Dossier de fichiers
xnio-provider.jar
Dossier de fichiers
adminps
91 Ko
Fichier WAR
barkbank
18 Ko
Fichier WAR
ejb2-container-jboss-beans
1 Ko
Document XML
ejb2-timer-service
3 Ko
Document XML
ejb3-connectors-jboss-beans
2 Ko
Document XML
ejb3-container-jboss-beans
1 Ko
Document XML
ejb3-interceptors-aop
27 Ko
Document XML
ejb3-timerservice-jboss-beans
1 Ko
Document XML
hdscanner-jboss-beans
2 Ko
Document XML
hsqldb-ds
6 Ko
Document XML
iiop-service
7 Ko
Document XML
jboss-local-jdbc
15 Ko
Archive WinRAR
jboss-xa-jdbc
15 Ko
Archive WinRAR
jca-jboss-beans
3 Ko
Document XML
jms-ra
85 Ko
Archive WinRAR
jmx-invoker-service
6 Ko
Document XML
jsr88-service
2 Ko
Document XML
legacy-invokers-service
3 Ko
Document XML
mail-service
2 Ko
Document XML
mysql-ds
2 Ko
Document XML

5 Déployer le projet sous Jboss en utilisant le fichier build.xml.

Pour déployer une nouvelle version de jsf, il est souvent nécessaire de :
. supprimer les répertoires tmp et work de petstore sous Jboss,
. vider les caches du navigateur,
. fermer le navigateur

La classe `MyPhaseListener` permet de logger les phases du cycle de vie d'une requête jsf.

TP13

Implémenter l'ejb stateless CustomerServiceBean.

- Spécifier la politique transactionnelle du bean, des méthodes, la data source utilisée.
- Utiliser la classe Junit CustomerService Test, pour vérifier les services de l'ejb.

Compléter les facelets createcustomer et updatecustomer.

Compléter la classe CustomerControllor.

Déclarer le bean managé dans le fichier faces-config

Définir les règles de natigation

Déployer le projet