TP 8 : Pages dynamiques avec JSP

Pascal GRAFFION
2013/12/05 17:50

Table des matières

TP 8 : Pages dynamiques avec JSP

Java Server Page, ou JSP, est une technologie basée sur Java qui permet aux développeurs de générer dynamiquement du code HTML, XML ou tout autre type de page web. C'est-à-dire qu'une page JSP (repérable par l'extension .jsp) aura un contenu pouvant être différent selon certains paramètres (des informations stockées dans une base de données, les préférences de l'utilisateur,...) tandis qu'une page web « classique » (dont l'extension est .htm ou .html) affichera continuellement la même information.

Les JSP résident dans un conteneur, tout comme les servlets. En fait, une JSP est un autre moyen d'écrire une servlet, c'est donc une classe Java dérivant de HttpServlet. Lorsqu'un utilisateur appelle une page JSP, le serveur web invoque le moteur de JSP qui crée un code source Java à partir du script JSP, compile la classe afin de fournir un fichier compilé (d'extension .class), c'est-à-dire qu'il constitue en fait une servlet à partir du script JSP.

Grâce à l'utilisation de balises, JSP permet d'intégrer facilement du code Java au sein du code HTML. L'intérêt principal de ce mécanisme par rapport aux servlets provient de la séparation entre la présentation (directement codée en HTML) et la logique applicative (traitements) fournie par Java. Il y a cinq types de balises :

Une directive (@) est une instruction insérée dans des tags HTML spéciaux.

```
<%@ include file="relativeURL" %>
```

Une déclaration (!) permet d'insérer du code directement dans la JSP. Elle peut être utilisée pour définir des attributs ou/et des méthodes à la sous-classe de HttpServlet générée.

```
<%! int variableDeClasse = 0; %>
```

Un scriptlet est utilisé pour placer du code dans la JSP. C'est généralement l'élément utilisé pour placer tout code Java, sauf les méthodes et les variables de classe.

```
<% int variable = 0;
out.println("On peut aussi écrire des variables : " + variable); %>
```

Une expression (=) sert à afficher du texte ou un résultat. Ce code est équivalent à un appel out.print(). Voici une variable :

```
<%= variable %>
```

Une action permet de fournir des instructions à l'interpréteur JSP.

```
<jsp:useBean id="customerDTO" class="com.dto.CustomerDTO"/>
```

En plus de ces balises, plusieurs variables sont disponibles dans une page JSP :

- out : le JSPWriter utilisé pour envoyer la réponse HTTP au client.
- page : la servlet elle-même.
- pageContext : une instance de PageContext qui contient les données associées à la page entière. Une page HTML donnée peut être passée entre plusieurs JSP.
- request : objet représentant la requête HTTP.
- response : objet représentant la réponse HTTP.
- session : la session HTTP, qui peut être utilisée pour conserver de l'information à propos d'un utilisateur d'une requête à une autre.

Hello PetStore !

Le code ci-dessous permet à une JSP d'afficher cinq fois la phrase « Hello Petstore ! » et la date du jour.

```
<%@ page import="java.util.Date"%>  // (1)
  <title>Hello PetStore!</title>
  <% Date today = new Date();%>  // (2)
  <%
     for (int i=0; i? i++) {
         out.println("Hello Petstore!<BR>");  // (3)
     }
  %>
  <BR>
  <center><%= today %></center>  // (4)
```

La directive import (1) a le même comportement que l'import java, dans notre cas, elle permet d'importer la classe java.util.Date qui est utilisée plus bas (2). Un scriptlet utilise une boucle for pour afficher la phrase « Hello Petstore ! ». Il est intéressant de constater que les scriptlets utilisent directement la variable out pour écrire et non System.out et que les balises

HTML sont interprétées (ici
 n'est pas affiché sur la page mais effectue plutôt un saut à la ligne). Enfin, une expression affiche (4) la date du jour.

Une fois packagé dans un fichier war (ant war) et déployé dans Tomcat (ant deploy), l'application est accessible; allez à l'adresse http://localhost:8080/hellops/hello.jsp pour exécuter la JSP :

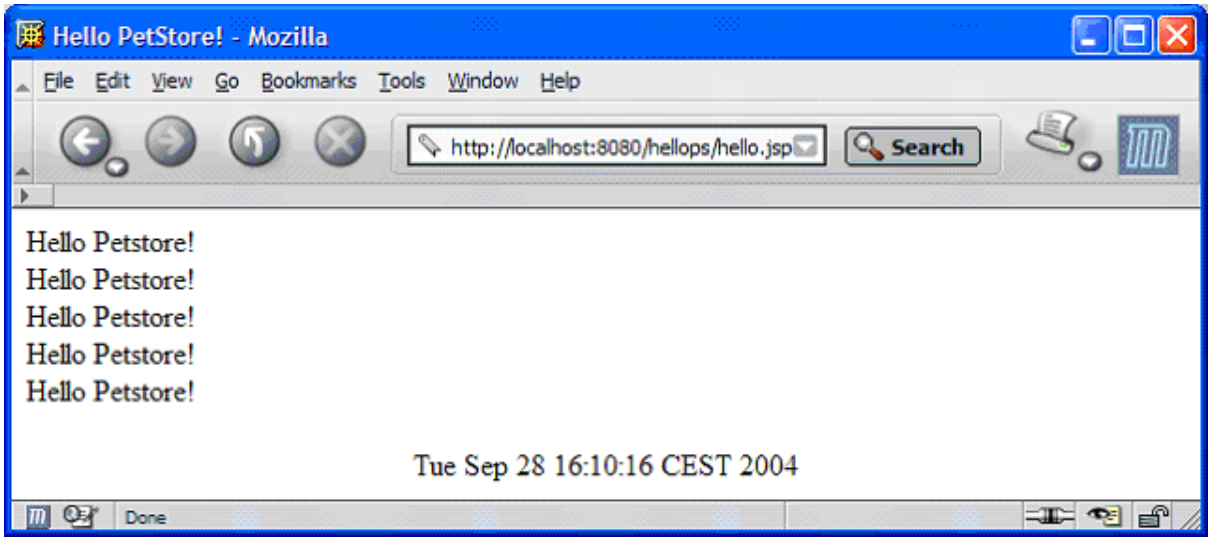


Figure 1 - Exécution de la JSP

Expression des besoins

Une grande part du chiffre d'affaires de YAPS est maintenant générée par les franchisés. Ces derniers sont globalement satisfaits de la possibilité de créer des clients en ligne, mais ils aimeraient avoir accès à plus d'informations. Ils veulent pouvoir consulter le catalogue des animaux domestiques en ligne. Pour que leurs clients puissent se rendre compte des animaux qu'ils cherchent à acheter, les franchisés voudraient avoir une image représentant chaque animal. Cela permettrait de montrer le contenu du catalogue aux clients. Pour cela, il faut que l'application web soit plus esthétique ainsi que simple à utiliser.

Vue utilisateur

Diagramme de cas d'utilisation

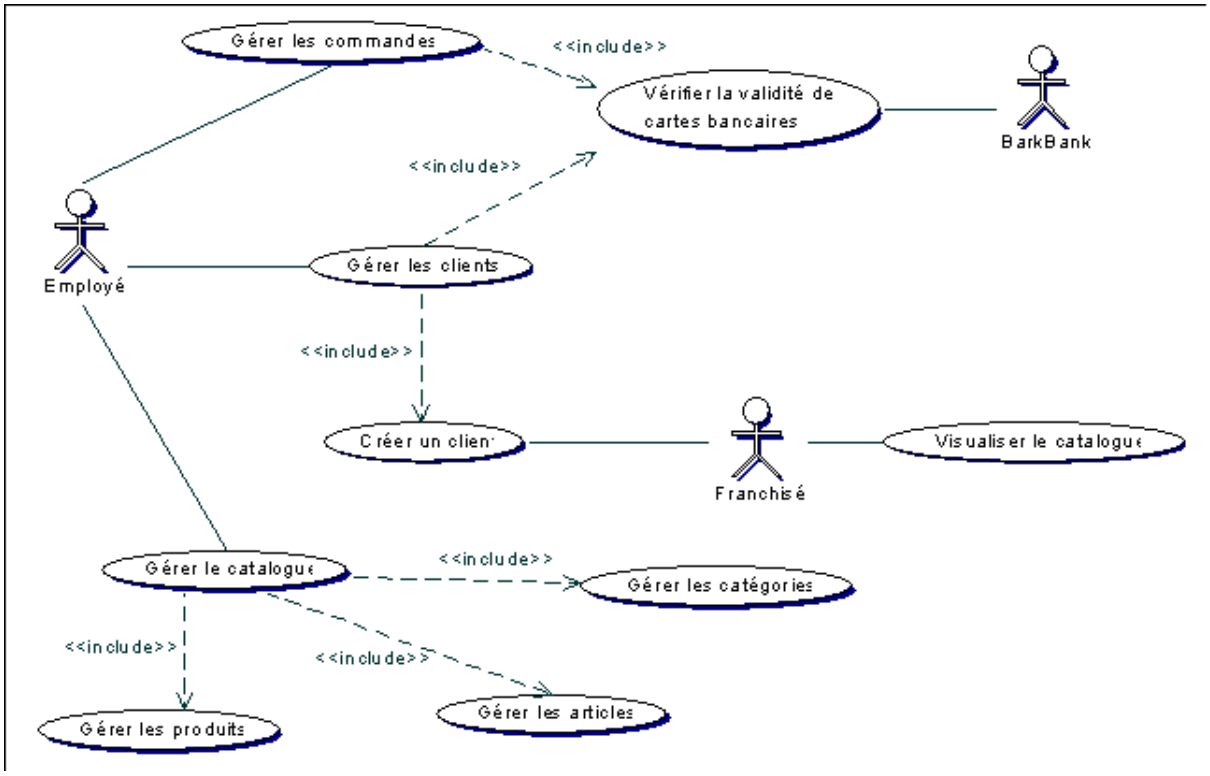


Figure 2 - Diagramme de cas d'utilisation

Cas d'utilisation « Visualiser le catalogue »

Nom	**Visualiser le catalogue.**
Résumé	Permet de visualiser le contenu du catalogue d'animaux domestiques.
Acteurs	Franchisé.
Pré-conditions	Aucune.
Description	Les franchisés veulent visualiser la totalité du catalogue des animaux domestiques. L'organisation de l'affichage doit être intuitive et suivre la démarche des clients : lorsqu'un client désire acheter un animal, il commence par donner la catégorie (« je voudrai acheter un chien ») puis le produit (« un caniche ») et enfin l'article lui-même (« un mâle adulte »). En face de chaque article une image devra être affichée représentant l'animal.À tout moment on veut pouvoir afficher les produits d'une catégorie différente.
Exceptions	((GLOBAL)) Si une erreur système se produit, l'exception FinderException doit être levée.

Diagramme d'activité

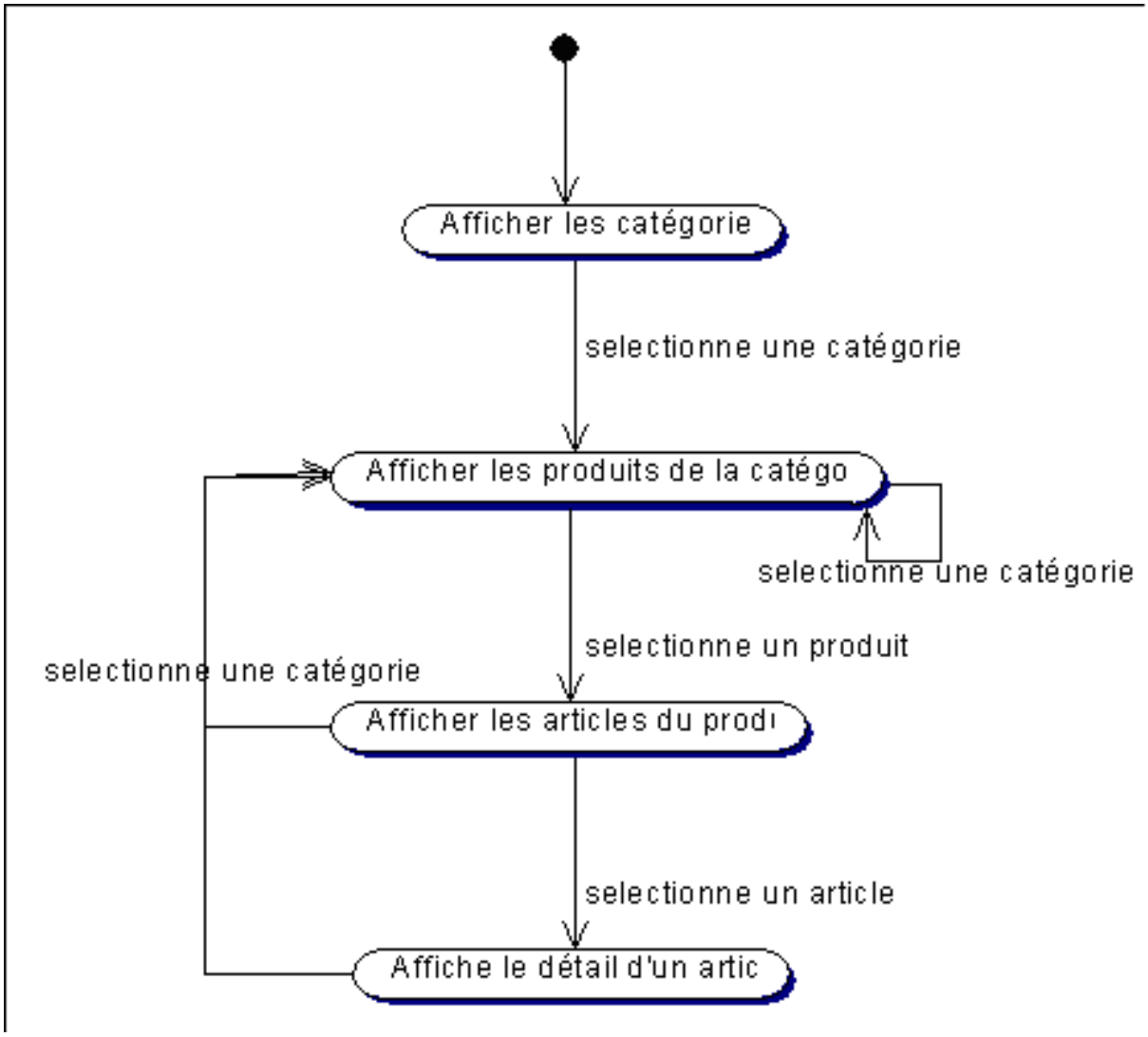
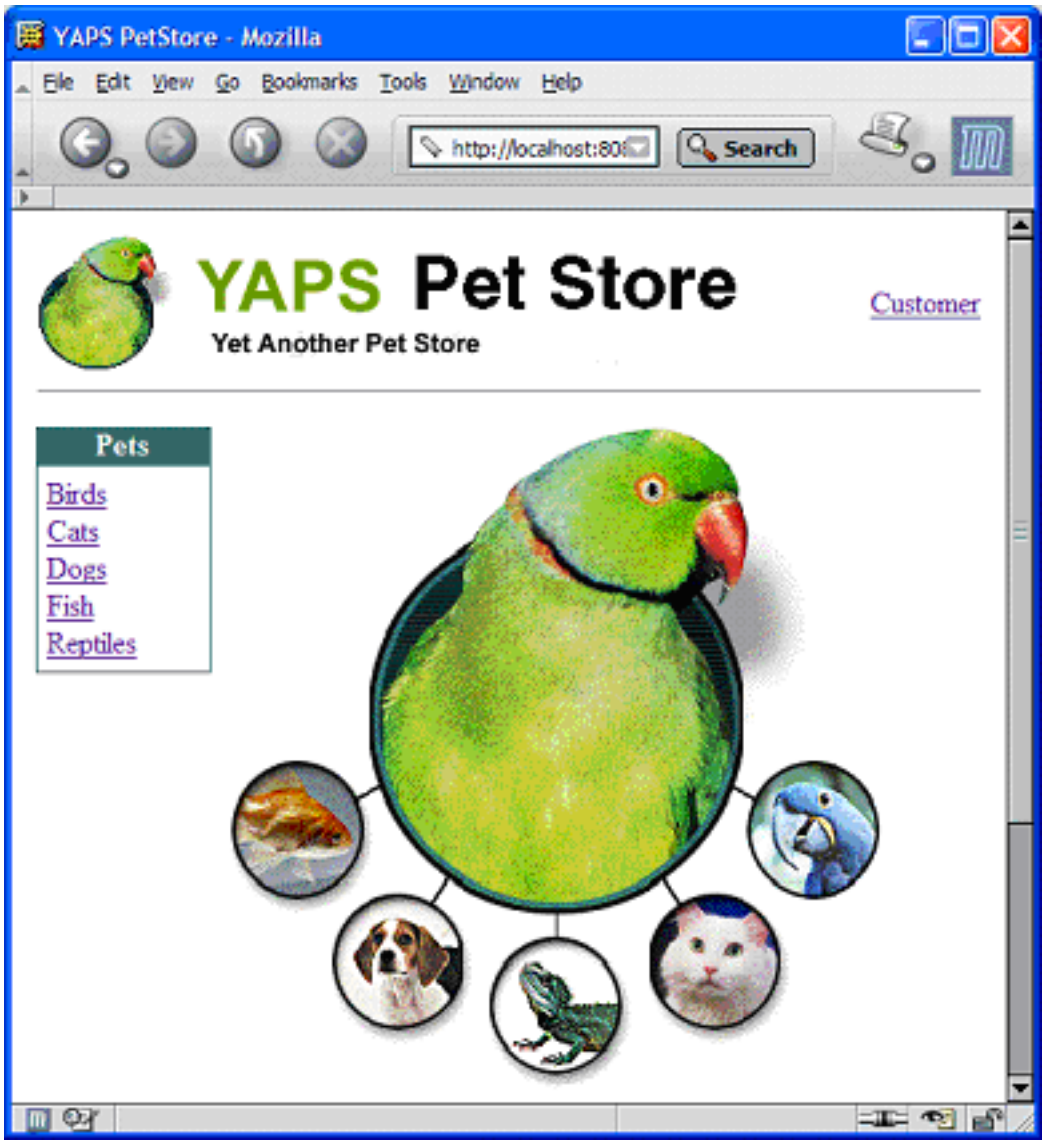


Figure 3 - Diagramme d'activité de la visualisation du catalogue

Le diagramme d'activité ci-dessus nous donne la représentation graphique des actions effectuées par les franchisés pour visualiser le contenu du catalogue. Il est à noter qu'à n'importe quel moment ils peuvent revenir à l'action « Afficher les produits de la catégorie ».

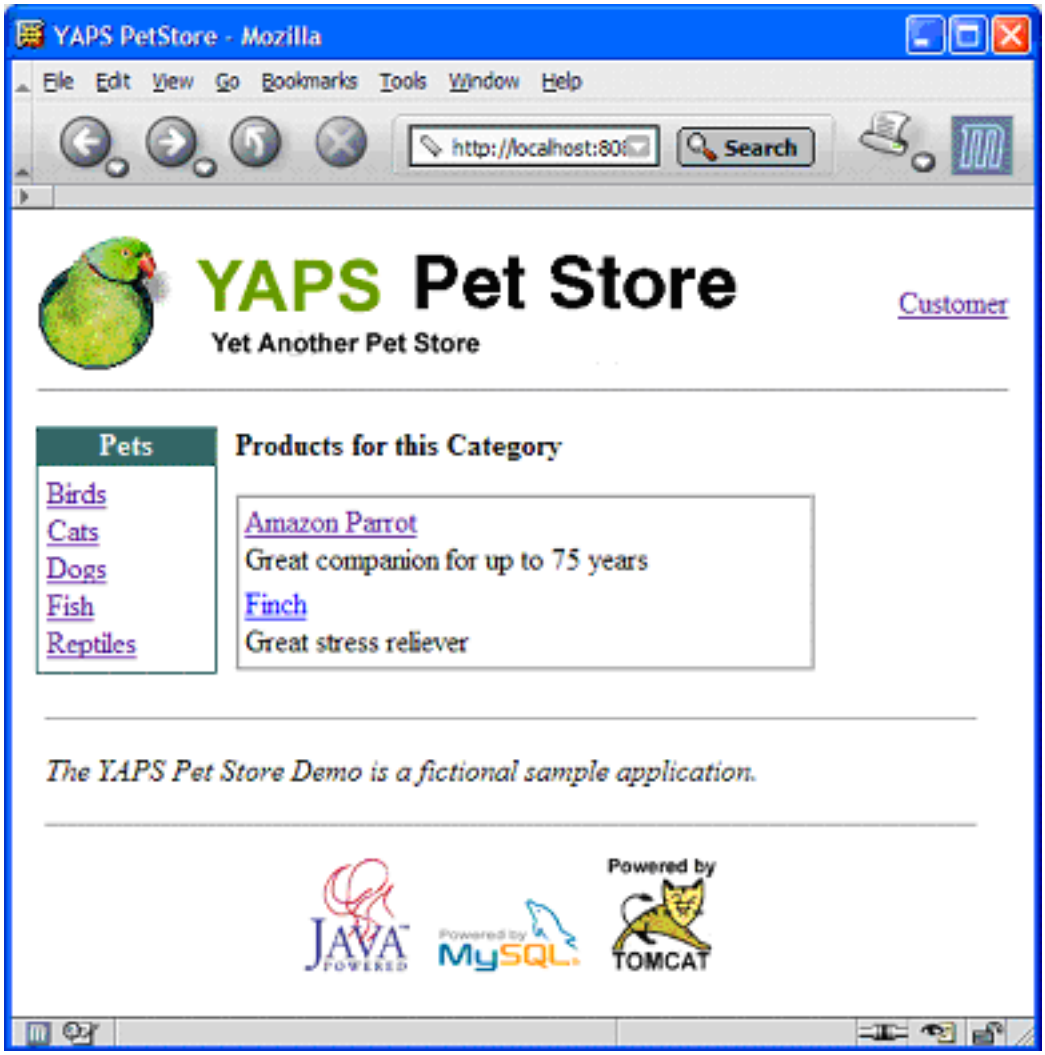
Ecrans



Les pages pour visualiser le catalogue sont construites des mêmes élements graphiques, c'est-à-dire :

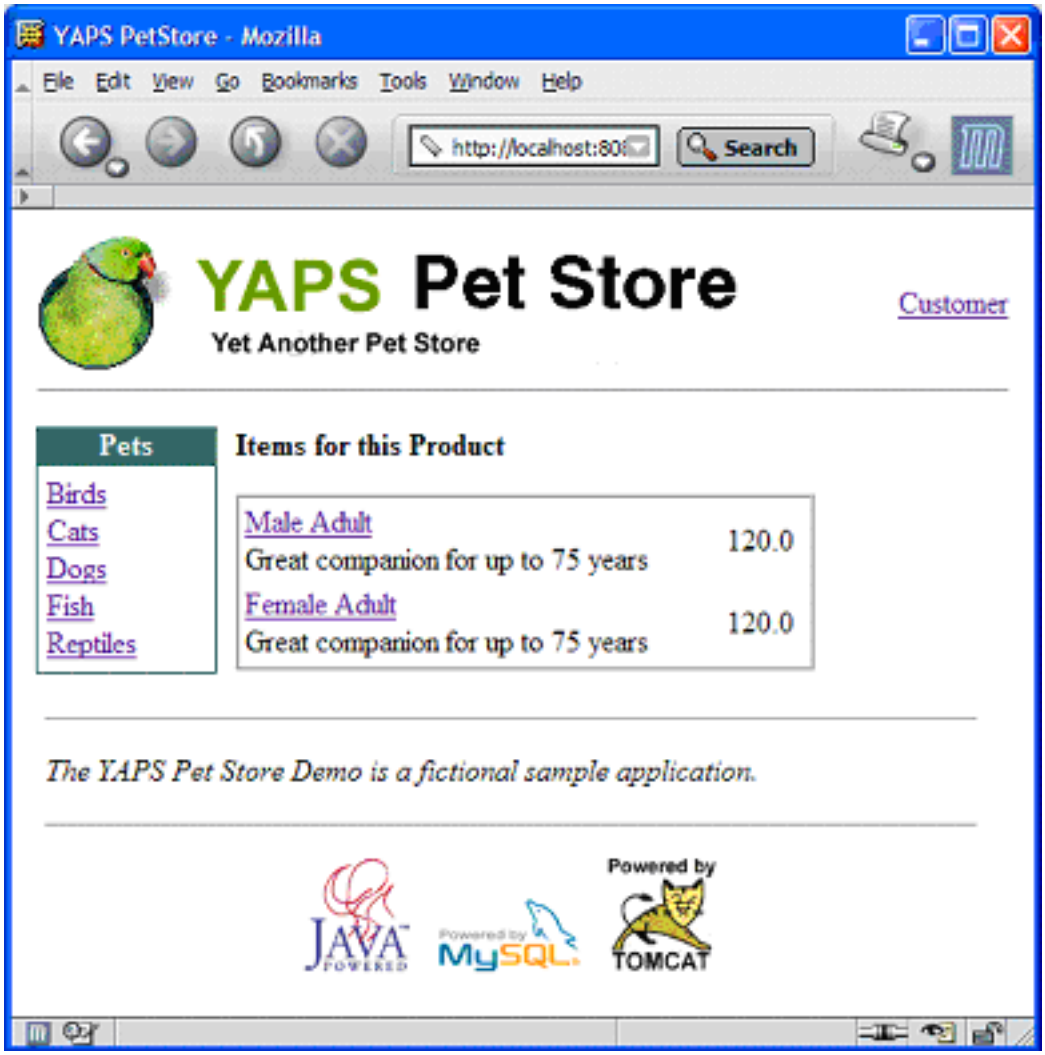
- L'en-tête affiche le logo et le nom de la société YAPS. En cliquant sur le logo, le franchisé est redirigé vers la page d'accueil. L'en-tête possède aussi un lien Customer qui permet d'accéder à la création d'un client.
- Sur la gauche, une barre de navigation permet au franchisé de sélectionner une catégorie. Il peut effectuer cette même action en cliquant sur la photo d'un animal.
- Et en pied de page se trouvent les logos des technologies utilisées (Java, MySQL et Tomcat).

En cliquant sur la catégorie Birds (Oiseaux) le franchisé est redirigé vers cette page qui affiche la liste des produits pour cette catégorie.



Pour chaque produit, on affiche son nom (Amazon Parrot) et sa description (Great companion for up to 75 years).

En cliquant sur le produit Amazon Parrot (Perroquet d'Amazone), le franchisé est redirigé vers la liste des articles. Dans notre cas, ce produit possède deux articles : Male Adult (adulte mâle) et Female Adult (adulte femelle).



Pour chaque article, on affiche son nom (Male Adult), son prix (120.0) ainsi que la description du produit (Great companion for up to 75 years).

Enfin, pour connaître le détail d'un article, il suffit au franchisé de cliquer sur son nom pour arriver sur cette page.

Le nom et le prix de l'article sont affichés ainsi que l'image représentant l'animal.

Le formulaire de création d'un client est le même hormis le fait qu'il est, lui aussi, décoré d'un en-tête (header.jsp), d'un bas de page (footer.jsp) et d'une barre de navigation (navigation.jsp).

Analyse et conception

Vue logique

Pour répondre aux besoins utilisateur, certaines de nos classes doivent être modifiées. Tout d'abord on s'aperçoit que les franchisés veulent connaître la liste des produits pour une catégorie donnée ainsi que la liste des articles pour un produit donné. Les méthodes que nous avions jusqu'à présent retournaient la liste complète des produits et des articles. Il nous faut donc garder ces fonctionnalités et en rajouter d'autres. C'est pour cela que dans le diagramme de classe ci-dessous vous voyez apparaître la méthode findProducts(String categoryId) au côté de findProducts() et findItems(String productId) ainsi que findItems().

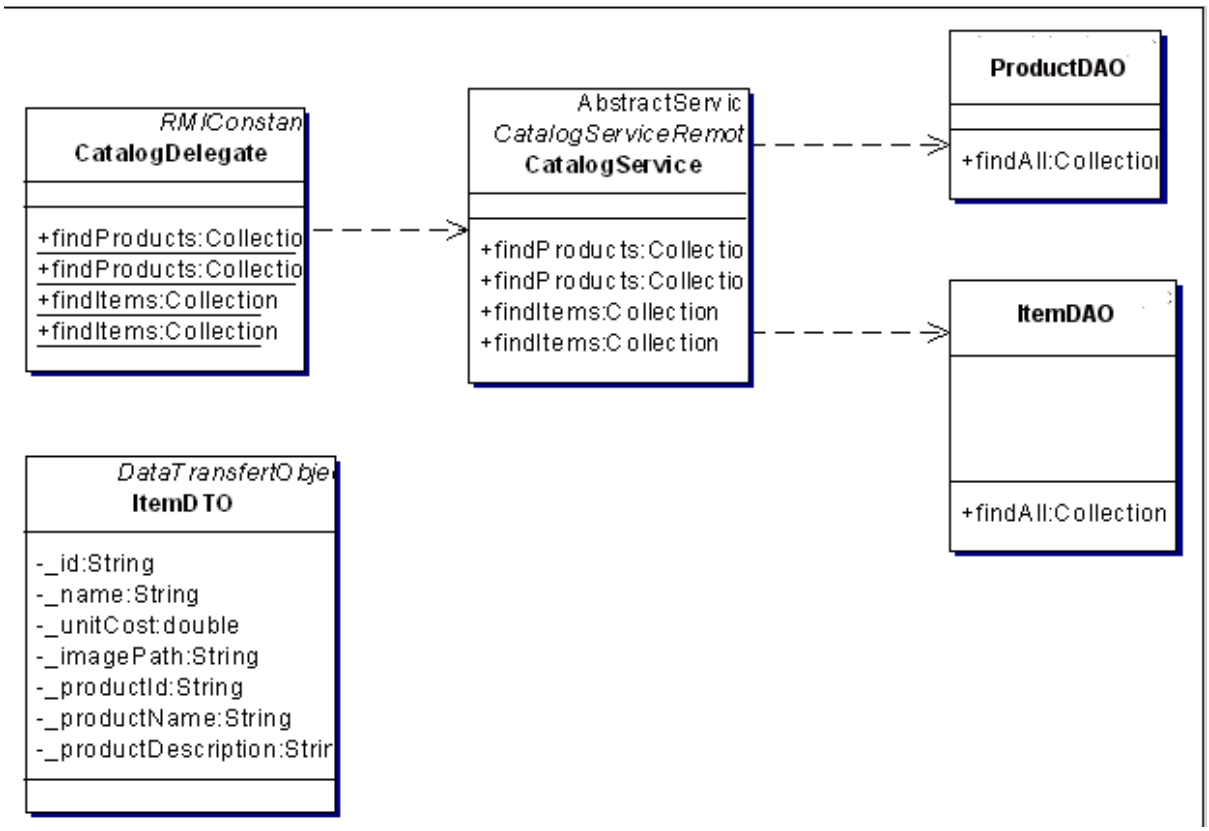


Figure 4 - Diagramme de classes avec les nouvelles méthodes findAll

Pour pouvoir afficher une image pour chaque animal, il faut avant tout stocker cette information. Deux possibilités s'offrent à nous. Soit on stocke l'image directement en base de données sous forme de BLOB (Binary Large Object), soit on stocke le nom du répertoire physique où se trouve l'image. Cette dernière solution est plus simple à mettre en oeuvre, il suffit de rajouter un nouvel attribut imagePath à la classe Item. Il faudra bien sûr aussi le rajouter dans ItemDTO ainsi que la description du produit qui est nécessaire dans les pages web.

Vue processus

Ci-dessous le diagramme de séquence illustrant la visualisation du catalogue. La page d'accueil, index.jsp appelle la servlet FindProductsServlet en lui passant en paramètre l'identifiant de la catégorie sélectionnée. Celle-ci invoque la classe CatalogDelegate pour obtenir une liste de ProductDTO qu'elle passe à la page products.jsp. Le même scénario se produit pour l'affichage de la liste des articles (items.jsp) ainsi que le détail (item.jsp).

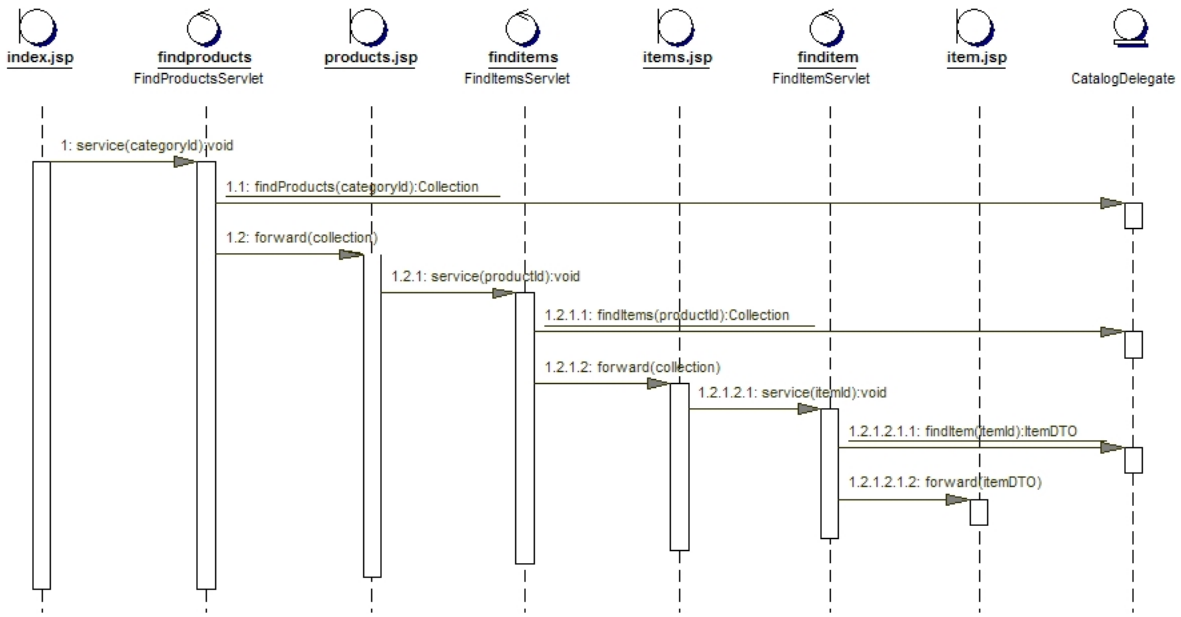


Figure 5 - Diagramme de séquence entre servlets et JSPs

Dans le chapitre 6, nous avions créé une page HTML statique (createcustomer.html) qui appelait une servlet pour effectuer le traitement. Cette servlet invoquait ensuite les objets métiers pour créer un client puis retournait une page statique confirmant la création. On s'aperçoit que dans ce modèle, la servlet est à la fois utilisée pour le traitement (appel serveur, gestion des exceptions...) que pour l'affichage (code HTML dans la servlet pour informer du succès de l'opération). Ce type d'architecture est appelé Model I, c'est-à-dire qu'une servlet ou une JSP est utilisée à la fois pour le traitement et la présentation.

Le problème majeur du Model I, est que le développement de pages dynamiques (soit par servlet soit par JSP) devient une tache complexe. En effet, il mêle deux compétences (web designer et programmeur) et deux langages distincts (HTML/ JavaScript et Java).

Dans le diagramme de séquence ci-dessus, on s'aperçoit que les JSP et les servlets effectuent un travail différent : la servlet exécute un traitement, appelle le serveur, gère les exceptions et renvoie le résultat de sa requête (sous forme de DTO) à la JSP. Celle-ci n'a plus qu'à afficher le contenu des Data Transfert Object. C'est le Model II, inspiré du design pattern MVC (Model-View-Controller), qui permet de décorréler traitement (servlet) et présentation (JSP).

Le design pattern MVC (Modèle-Vue-Contrôleur) est plus particulièrement utilisé dans la création d'interfaces graphiques. Inventé par la communauté Smalltalk, il a été ensuite repris dans plusieurs langages et a inspiré le développement de l'API Swing. Il permet d'organiser une application en trois composants principaux :

- Un modèle qui correspond aux données de l'application (ici les objets DTO).
- Une vue qui correspond à la présentation visuelle de l'application (la JSP).
- Un contrôleur qui définit l'état de la vue en fonction des données gérées par le modèle (la servlet).

Les paramètres passés entre une serlvet et une JSP ne suivent pas la convention du langage Java. En effet, en Java, nous sommes habitués à appeler des méthodes auxquelles nous passons des paramètres typés. Les servlets héritant de HttpServlet, elles ont des méthodes et des paramètres figés, comme par exemple service(HttpServletRequest, HttpServletResponse). C'est justement en utilisant l'objet HttpServletRequest que les JSP et les servlets arrivent à échanger des données.

Affichage de la liste des produits

Le code ci-dessous permet à la page products.jsp d'afficher la liste des produits sous forme de lien. Cliquez sur un produit et vous serez dirigé vers la servlet findItems en passant au travers du paramètre productId l'identifiant du produit. Le nom du produit sera surligné indiquant aux franchisés l'endroit où ils doivent cliquer (3). Notez la présence de l'expression %= request.getContextPath() %>, celle-ci a pour effet de préfixer l'URL par le chemin de l'application, dans notre cas petstore.

```
<A href="<%= request.getContextPath() %>/finditems?productId=<%= productDTO.getId() %>">
<%= productDTO.getName() %>
</A>
```

Lors de l'exécution de cette page le code jsp sera évalué et affichera le résultat (au format HTML) suivant :

```
<A href="/petstore/finditems?productId=AVCB01">Amazon Parrot</A>
```

Affichage du détail d'un article

Pour afficher le détail d'un article, la servlet FindItemServlet récupère l'identifiant de cet article (1) au travers de la requête et accède au serveur (2) qui lui renvoie un objet ItemDTO. Cet objet est ensuite stocké dans cette même requête (3) puis passé à la JSP (4). Cette dernière n'aura plus qu'à récupérer l'objet itemDTO stocké en utilisant request.getAttribute("itemDTO").

```
protected void service(final HttpServletRequest request, final HttpServletResponse response) throws ServletException,
IOException {
  final ItemDTO itemDTO;
   String itemId = request.getParameter("itemId");  // (1)
  try {
     // Gets the item
     itemDTO = CatalogDelegate.findItem(itemId);  // (2)
     // ou CatalogDelegateFactory().createCatalogDelegate().findItem(itemId);  // (2)
     // puts the item into the request
     request.setAttribute("itemDTO", itemDTO);    // (3)
     // Goes to the item page passing the request // (4)
     getServletContext().getRequestDispatcher("/item.jsp").forward(request, response);
  (...)
}
```

Comme vous pouvez le voir ci-dessus, la servlet gère les enchaînements en utilisant le RequestDispatcher avec, comme paramètre, la destination à atteindre. Dans notre cas, lorsque la FindItemServlet a trouvé l'article (2) en question, elle redirige l'appel vers la page item.jsp. Pour une JSP, l'enchaînement se fait différemment. Rappelez-vous que le rôle d'une JSP est de fournir du code interprétable par un navigateur, c'est-à-dire du HTML. Un lien doit donc être fait en utilisant les balises <a href="...">.

La page item.jsp (la Vue) contient un minimum de code Java, manipulant l'instance d'ItemDTO passée en paramètre.

```
<%@ page errorPage="error.jsp"%>
<%@ page import="com.yaps.petstore.common.dto.ItemDTO"%>
<html>
...
    <%--CENTRAL BODY--%>
      <%
        final ItemDTO itemDTO = (ItemDTO) request.getAttribute("itemDTO");
      %>
      <P><strong><%= itemDTO.getName() %></strong></P>
...
```

Affichage des erreurs

Pour homogénéiser les pages du site, la servlet qui traitait les erreurs (ErrorServlet) est transformée en JSP comportant les mêmes éléments décoratifs (en-tête, pied de page et navigation) que les autres pages. Pour rendre la gestion des erreurs plus robuste, on peut utiliser le mécanisme automatique mis en place pour les JSP. Il suffit de déclarer la directive < % @page isErrorPage="true" %> dans la page error.jsp et < % @page errorPage="error.jsp" %> dans toutes les autres. Ceci a pour effet de rediriger automatiquement l'utilisateur vers la page error.jsp en cas d'exception non contrôlée.

Vue implementation

Les pages JSP et les images se trouvent dans le répertoire Yasp/resources. Les servlets, quant à elles, sont dans le paquetage com.yaps.petstore.web.servlet.

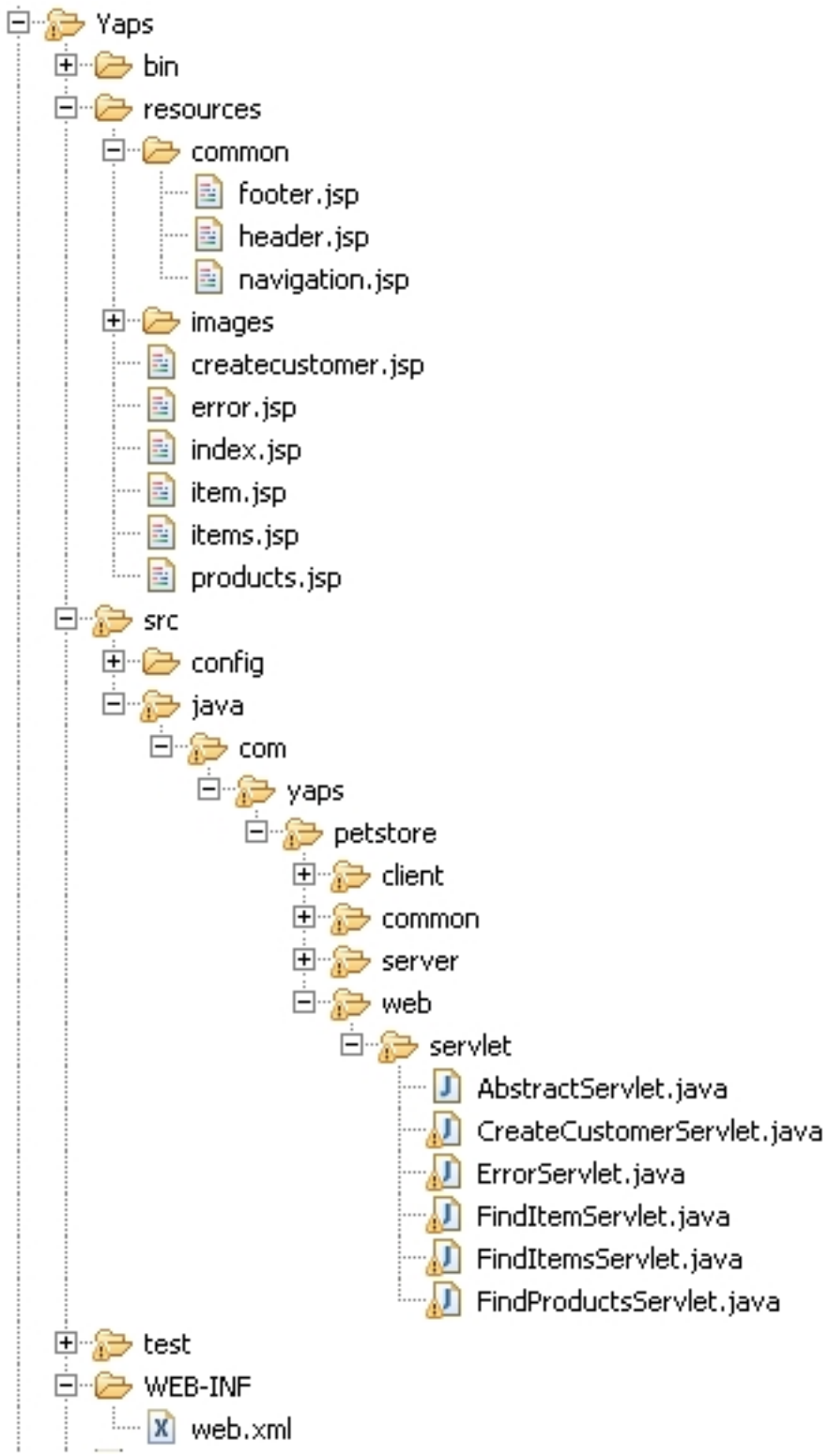


Figure 6 - Répertoires contenant les JSPs

Les pages représentant l'en-tête, le bas de page et la barre de navigation se trouvent dans le répertoire common. Elles sont utilisées par les autres pages grâce à la directive include des JSP. Ci-dessous le patron d'une page qui nous montre comment utiliser la gestion des erreurs (1) et l'inclusion des pages décoratrices (2). Ensuite, il suffit de rajouter le code spécifique à chaque page dans la partie centrale (3).

```
<%@ page errorPage="error.jsp"%>            // (1)
  <title>YAPS PetStore</title>
<table cellspacing="0" cellpadding="5" width="100%">
```

```
  <%--HEADER--%>
  <tr>
    <td colspan="3">
      <jsp:include page="common/header.jsp"/>      // (2)
    </td>
  </tr>
  <tr>
    <%--NAVIGATION--%>
    <td valign="top" width="20%">
      <jsp:include page="common/navigation.jsp"/>  // (2)
    </td>
    <td align="center" width="60%">
    <%--CENTRAL BODY--%>
     (...)              // (3)
  <%--FOOTER--%>         // (2)
  <tr>
    <td colspan="3">
      <jsp:include page="common/footer.jsp"/>
    </td>
  </tr>
</table>
```

Architecture

Le diagramme de composants ci-dessous nous montre comment s'insère le nouveau sous-système JSP. Il est invoqué par le navigateur via HTTP, exécuté côté serveur et dialogue ensuite avec les servlets.

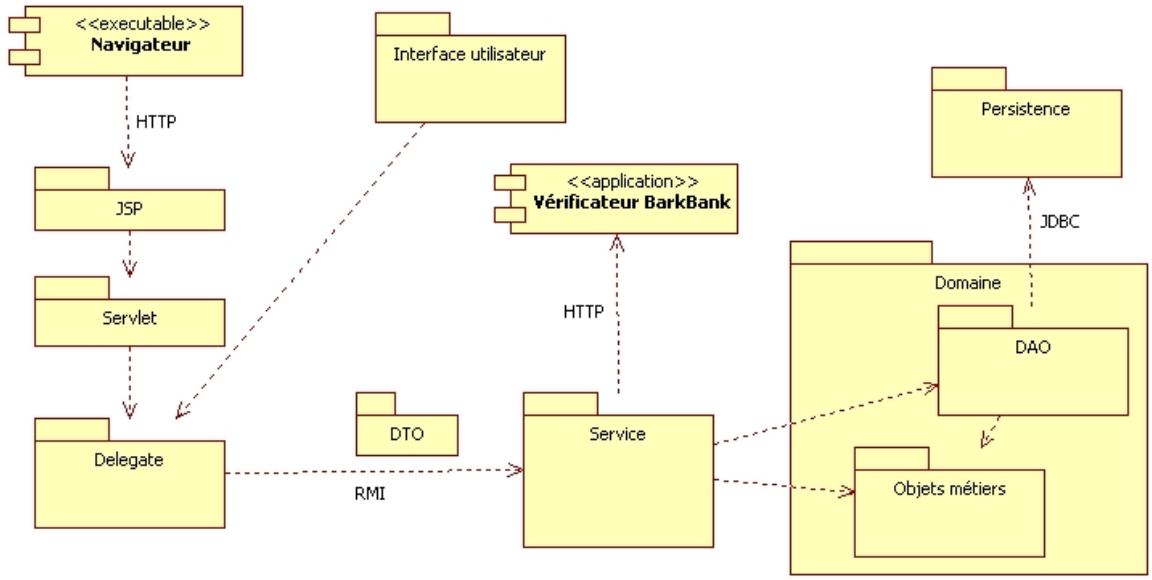


Figure 7 : Diagramme de composants avec servlet et JSP

Schéma de la base de données

Pour permettre l'affichage d'une image par animal, la table t_item se voit enrichie d'une nouvelle colonne : imagePath. La tache Ant yaps-create-db du fichier build.xml mettra à jour automatiquement cette table.

```
mysql> desc t_item;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id          | varchar(10)  |      | PRI |         |       |
| name        | varchar(50)  |      |     |         |       |
| description | varchar(255) | YES  |     | NULL    |       |
| unitCost    | double       |      |     | 0       |       |
| product_fk  | varchar(10)  |      | MUL |         |       |
```

```
| imagePath   | varchar(255) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
```

Vue déploiement

Le diagramme de déploiement est le même que celui de la précédente version. L'application se décompose donc en un client lourd (Petstore Client), un client léger (Petstore Web) est un serveur central (Petstore Server) traitant les demandes des employés et des franchisés et dialoguant avec un acteur externe, la BarkBank.

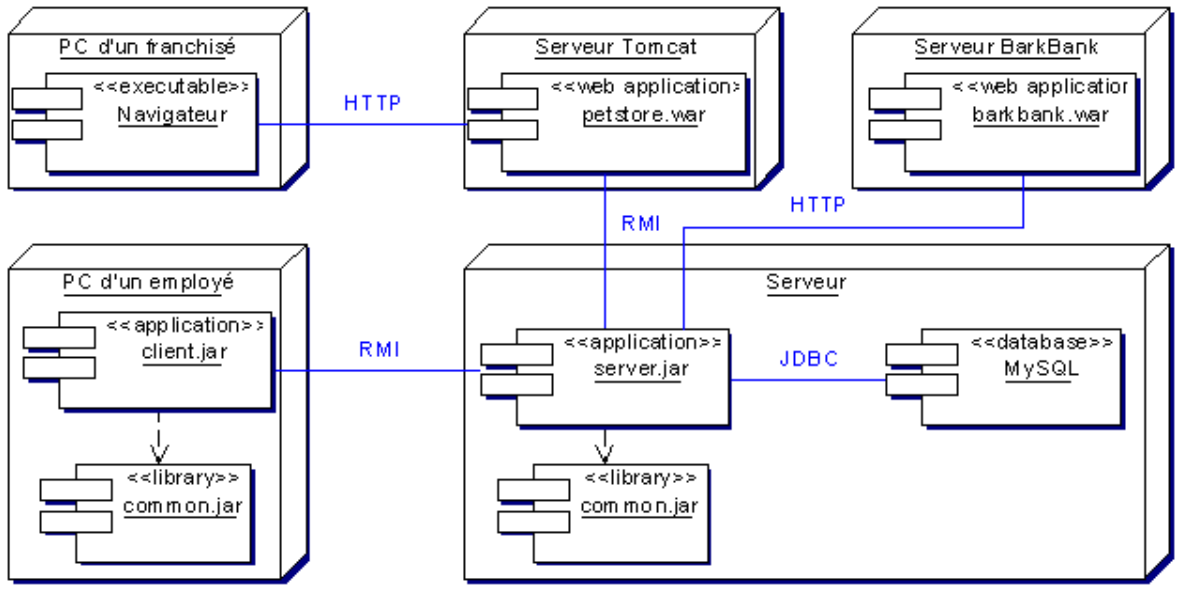


Figure 8 - Diagramme de déploiement

Suite aux améliorations effectuées sur la partie web, le contenu du fichier petstore.war a été enrichi. Les servlets se trouvent toujours sous le répertoire WEB-INF/classes alors que les JSP et les images se trouvent dans leurs sous-répertoires respectifs situés à la racine.

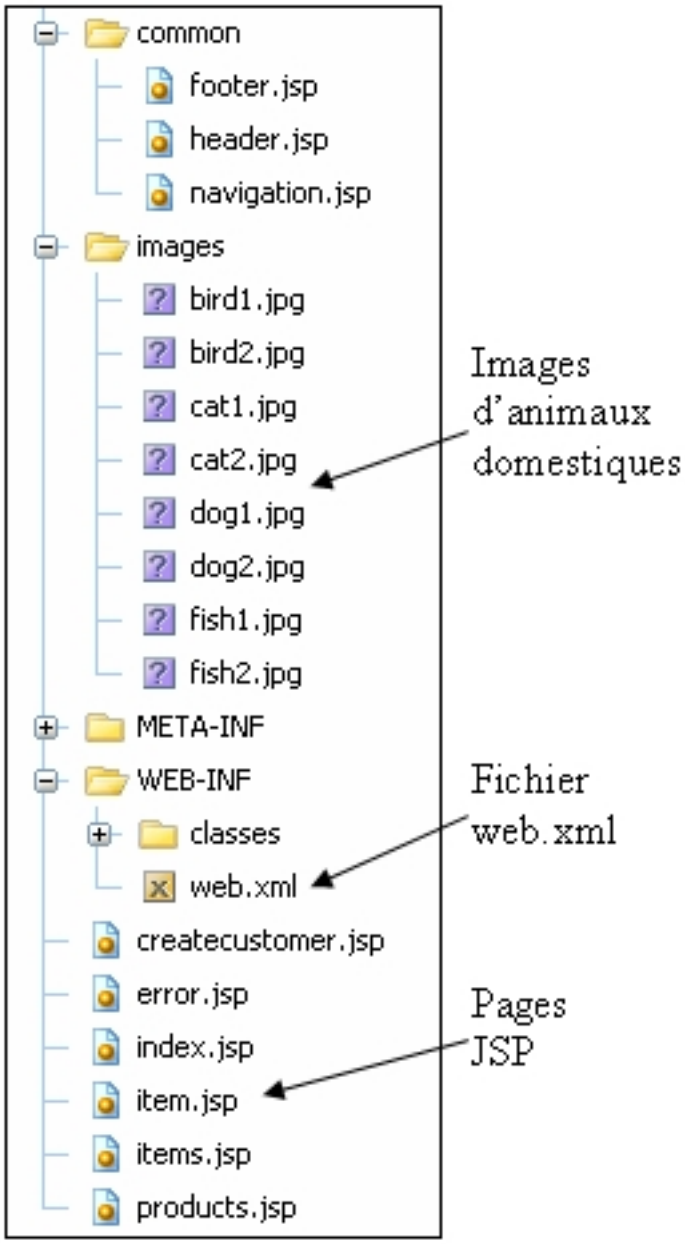


Figure 9 - Contenu du fichier petstore.war

Le fichier web.xml de l'application Petstore Web ne possède aucune information concernant les pages JSP. Par contre, on y liste toutes les servlets ainsi que leur alias:

Servlet	Alias
CreateCustomerServlet	createcustomer
FindItemServlet	finditem
FindItemsServlet	finditems
FindProductsServlet	findproducts

Implémentation

Vous pouvez maintenant développer l'application à partir de la version précédente ou télécharger la liste des classes fournies pour commencer votre développement. Les paquetages du client vous sont entièrement donnés et il ne manque que la classe CatalogService dans la partie serveur. Concernant la partie web, il vous faudra développer certaines des JSP et des servlets pour répondre aux besoins des utilisateurs.

Recette utilisateur

La classe AllTests appelle désormais la classe VisualiseCatalogTest qui, grâce à HTTPUnit simulera les actions des franchisés lors de la consultation du catalogue. Par exemple, le code ci-dessous se positionne sur la page d'accueil (1) et clique sur le premier lien qu'il rencontre qui se nomme findproducts?categoryId (2). Cette action a pour effet d'appeler la servlet FindProductsServlet et d'arriver sur la page products.jsp. Ensuite, un clic est effectué sur cette même page qui a pour effet d'appeler la FindItemsServlet (3). Enfin, un dernier clic sur la page items.jsp permet d'arriver sur la dernière page item.jsp (4) qui affiche le détail d'un article.

```
public void testVisualiseCatalog() throws Exception {
  (...)
  // The test starts at the index.jsp page
  indexJSP = webConversation.getResponse(URL_PETSTORE + "/index.jsp");  // (1)
  // We click on the first link of the index.jsp page
  productsLink = getTheFirstWebLinkThatMatches(indexJSP, "findproducts?categoryId");  // (2)
  productsJSP = productsLink.click();
  // We click on the first link of the products.jsp
  itemsLink = getTheFirstWebLinkThatMatches(productsJSP, "finditems?productId");  // (3)
  itemsJSP = itemsLink.click();
  // We click on the first link of the items.jsp
  itemLink = getTheFirstWebLinkThatMatches(itemsJSP, "finditem?itemId");   // (4)
  itemLink.click();
}
```

Les tests de recette peuvent aussi être exécutés grâce aux scénarios de tests fournis dans le répertoire Yaps/test/selenium.

Pour cela, lancez Selenium IDE depuis Firefox (Item "Selenium IDE" du menu Outils) puis ouvrez la suite de tests Yaps/test/selenium/testSuite.html pour la rejouer.

Résumé

Cette version de l'application permet à un franchisé de consulter la totalité du catalogue de la société YAPS en ligne. Les pages sont dites dynamiques car leur contenu change en fonction des données. Le langage HTML permet de construire des pages statiques, les servlets de traiter les requêtes HTTP et les JSP de gérer le dynamisme de l'affichage. Comme nous l'avons vu au chapitre six, une servlet peut évidemment être utilisée pour afficher une page dynamique (Model I). Le problème avec cette architecture est que les langages Java et HTML cohabitent dans la même classe, ce qui rend difficile à développer ou à déboguer. JSP est la technologie idéale lorsqu'on a besoin de développer des pages dynamiques.

Références

JavaServer Pages Technology http://www.oracle.com/technetwork/java/javaee/jsp/

The JSP Resource Index http://www.jspin.com/

Servlets and JavaServer Pages
Jayson Falkner, Kevin Jones. Addison-Wesley. 2003.

JavaServer Pages
Hans Bergsten. O'Reilly. 2003.

Understanding JavaServer Pages Model 2 architecture http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

Model View Controller http://c2.com/cgi/wiki?ModelViewController

HttpUnit http://httpunit.sourceforge.net/

Selenium IDE http://seleniumhq.org/docs/02_selenium_ide.html