Rapport Projet : INF-5101B
Framework d'application d'entreprise

Guillaume COUVOUTE
Yoann IBORRA

Sommaire :

Contexte de notre site de e-commerce

Notre site de e-commerce est destiné à mettre en relation des fabriquants de produits technologiques (Ordinateur, portable, smartphone, …) et des acheteurs.

Notre site est un une application entreprise, contenant un module EJB et un module WAR.

Il est aussi multi-langue (français, anglais).

Il existe deux types de compte, celui destiné aux fabriquants qui permet de rajouter ou supprimer des produits du catalogue, et celui destiné aux clients qui peuvent se connecter, remplir et vider un panier, enregistrer leur carte bancaire.

En plus de ces deux types de compte, il en existe en premier qui représente l'administrateur du site e-commerce. Cette personne peut donc modifier tous le contenu du site en ce qui concerne les contenus des tables SQL

Structure du projet

Notre projet se découpe en plusieurs parties, on peut en distinguer trois parties principales, celle du Java EE pur écrit en Java, celle des pages HTML sans formulaire écrites en JSP et celle des pages HTML avec formulaire écrites en JSF .

Tous ce qui concerne les beans et les controlleurs qui leurs sont associés, ils se situent dans le module EJB

Tous le reste se situe dans le module WAR :

- Le dossier servlet qui contient la servlet controlleur de notre application
- Le dossier handler qui permet de faire les traitements de l'application suivant l'URL, un URL correspond à un unique handler.
- Le dossier ManagedBean qui contient les Managed Bean necessaires à la partir JSF de notre projet.
- Le dossier fonction qui contient plusieurs fonctions permettant d'afficher certaines données ou d'autres suivant la langue utilisée.
- Le dossier ressources qui contient tous les fichiers properties des bundle nécessaires à l'utilisation de plusieurs langues (en l'occurence dans notre site, le français et l'anglais)

La partie HTML JSP comme la partie HTML JSF se situent dans le module WAR, et plus précisément dans le dossier web.

Les pages JSP de la partie HTML JSP sont dans le sous dossier WEB-INF donc l'accès est interdit à l'utilisateur tandis que la partie HTML JSF n'est pas dans un sous dossier car vu que l'on passe par la servlet JSF, nous ne pouvons pas rediriger les requêtes comme nous le voulons.

Partie HTML JSP :

- Les Dossiers web/js et web/css contiennent les fichiers js et css nécessaires à toutes les JSP
- Le dossier web/images qui contient les images de l'application
- Le dossier web/WEB-INF qui contient les pages JSP en plus des fichiers de configuration de l'application

Partie HTML JSF :

- Le dossier web/ressources qui contient les fichiers js et css nécessaires à toutes les JSF
- Le dossier web/images qui contient les images de l'application
- Le dossier web/admin qui contient toutes les pages JSF réservés à l'administrateur. La restriction à l'accès à ces pages se fait par le moyen d'un filtre appliqué à la servlet de JSF

Description de l’interface côté client

Pour ce qui de la partie HTML, CSS et Javascript, nous avons utilisé le Framework Bootstrap. Une partie des interactions entre le navigateur et le serveur ce font en AJAX au travers de la bibliothèque jQuery, l’AJAX nous en grande majorité pour le panier du client, c’est à dire que lorsqu’il souhaite rajouter un article dans son panier, il n’a pas besoin de recharger la page, il se contente d’envoyer le produit à rajouter puis actualise le DOM correspondant à l’affichage du panier.

```
var cart = {
    'add': function(product_id, quantity) {
        $.ajax({
            url: '/ProjectEJB-war/web/addBasket?id='+product_id,
            type: 'post',
            data: 'product_id=' + product_id + '&quantity=' + (typeof(quantity) != 'undefined' ? quantity : 1),
            dataType: 'json',
            beforeSend: function() {
                $('#cart > button').button('loading');
            },
            success: function(json) {
                $('.alert, .text-danger').remove();

                $('#cart > button').button('reset');

                if (json['success']) {
                                    json['total'] = Math.round10(json['total'],-2);
                                    $inter="";

                                    if(json['lang'] === "fr"){
                                        if(json['item']===1) $inter = "1 object - "+json['total']+" &euro;";
                                        else $inter = json["item"]+" objects - "+json['total']+" &euro;";
                                    }else if(json['lang'] === "en"){
                                        if(json['item']===1) $inter = "1 item - $"+json['total'];
                                        else $inter = json["item"]+" items - $"+json['total'];
                                    }
                                    $truc = "<button type=\"button\" data-loading-text=\"Loading...\" class=\"btn btn-inverse
                                    btn-block btn-lg\"><i class=\"fa fa-shopping-cart\"></i> <span id=\"cart-total\">"+$inter+"</span>
                                    </button>";
                                    $("#cart").html($truc);
                                    $('html, body').animate({ scrollTop: 0 }, 'slow');
                }
            }
        });
    },
```

Nos différentes JSP utilisent la bibliothèque Core du JSTL, comme par exemple la balise *<c:import>* qui nous permet d'inclure d'autres JSP dans une JSP, ceci évite la recopie de code identique entre les différentes JSP. L'utilisation de la balise *<c:out>* évite les failles de sécurités XSS.

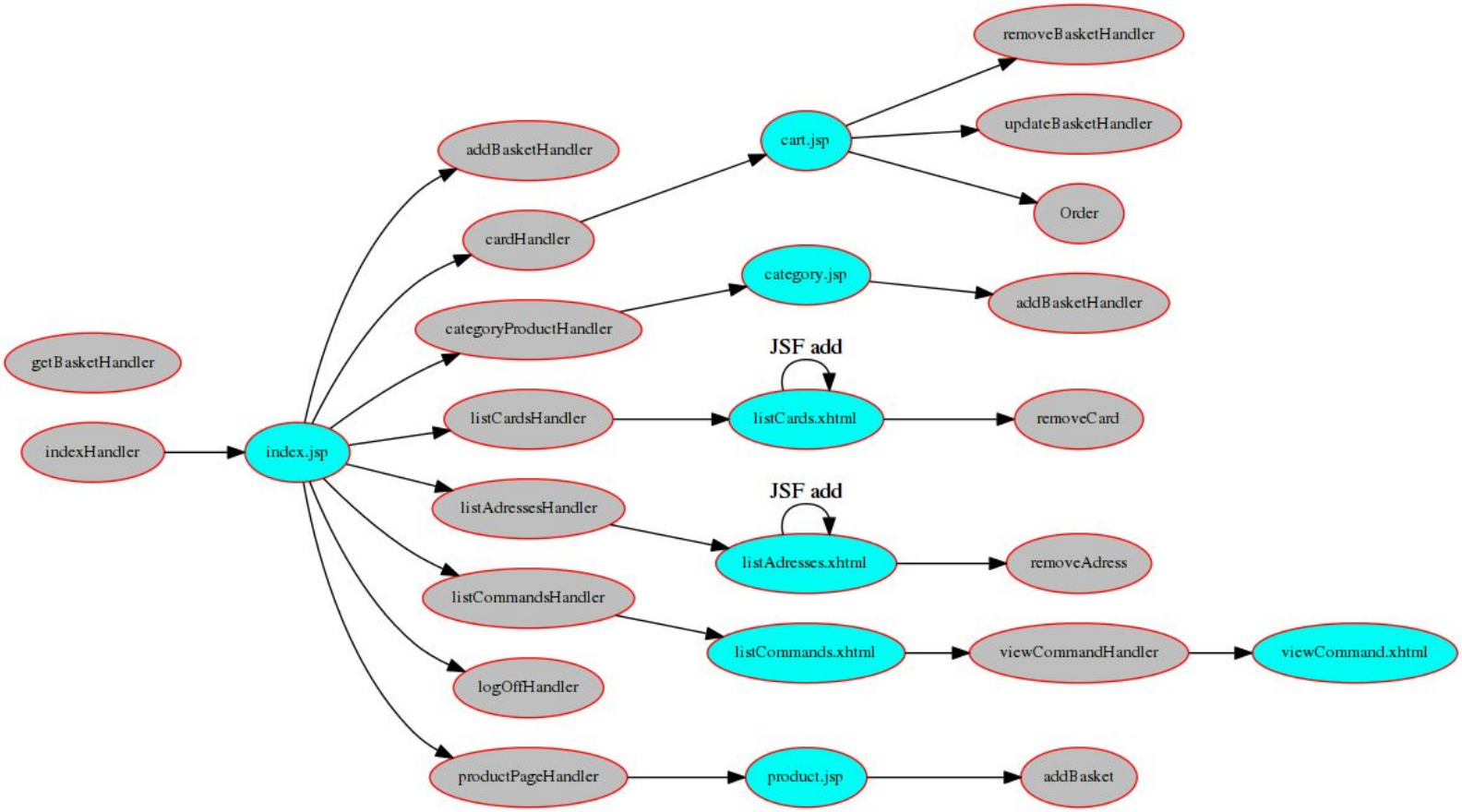
```jsp
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html lang="fr">

<head>
    <c:import url="/WEB-INF/inc/head.jsp" />
    <title>My Shop</title>
</head>

<body class="common-home">

    <c:import url="/WEB-INF/inc/navtop.jsp" />

    <c:import url="/WEB-INF/inc/header.jsp" />

    <div class="container">
        <c:import url="/WEB-INF/inc/navbar.jsp" />
    </div>

    <div class="container">
        <div id="content" class="row">
            <div class="col-sm-12">
                <div id="slideshow0" class="carousel slide" data-ride="carousel">
                    <!-- Indicators -->
                    <ol class="carousel-indicators">
                        <li data-target="#slideshow0" data-slide-to="0" class="active"></li>
                        <li data-target="#slideshow0" data-slide-to="1"></li>
                    </ol>

                    <!-- Wrapper for slides -->
                    <div class="carousel-inner" role="listbox">
                        <div class="item active"><img src="/project/images/MacBookAir-1140x380.jpg" alt=""></div>
                        <div class="item"><img src="/project/images/iPhone6-1140x380.jpg" alt=""></div>
                    </div>

                    <!-- Controls -->
                    <a class="left carousel-control" href="#slideshow0" role="button" data-slide="prev">
                        <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
                        <span class="sr-only">Previous</span>
                    </a>
                    <a class="right carousel-control" href="#slideshow0" role="button" data-slide="next">
                        <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
                        <span class="sr-only">Next</span>
                    </a>
                </div>
            </div>
        </div>
```

Schéma de Navigation

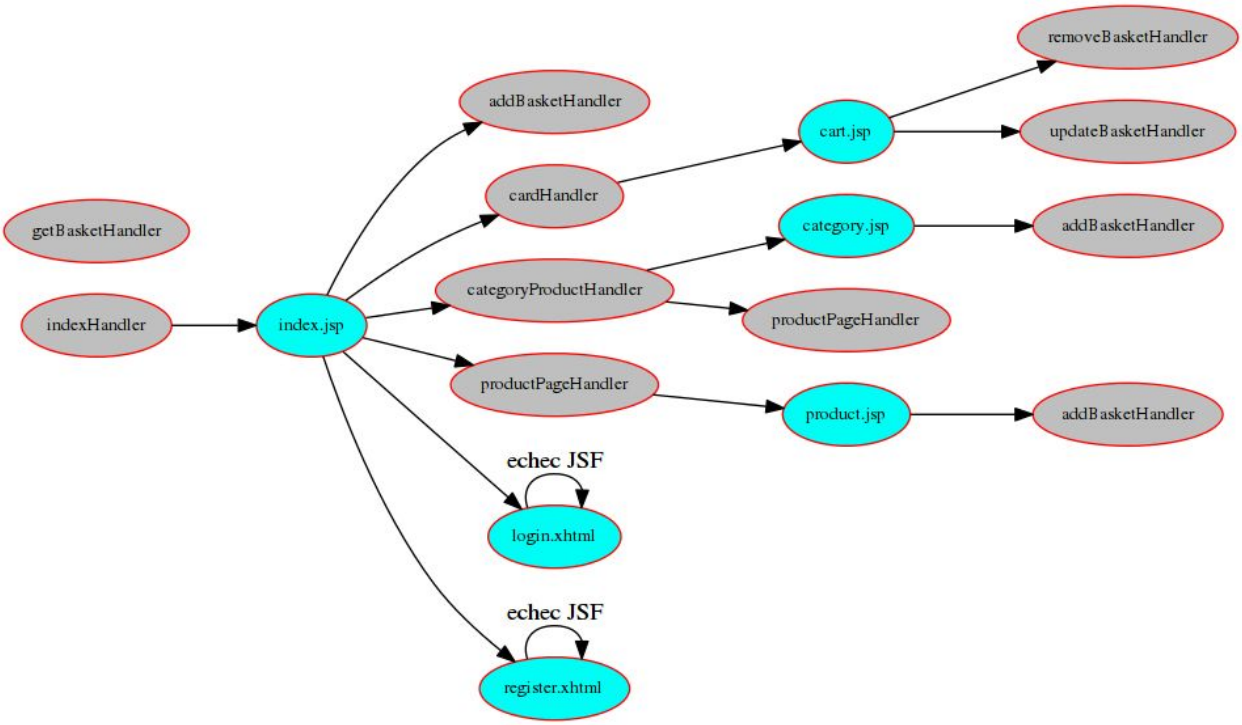
Voici nos schémas de navigation sachant que le handler getBasketHandler est gérer par du jQuery et qu'il est appelé par ce dernier à chaque page.

Partie Order

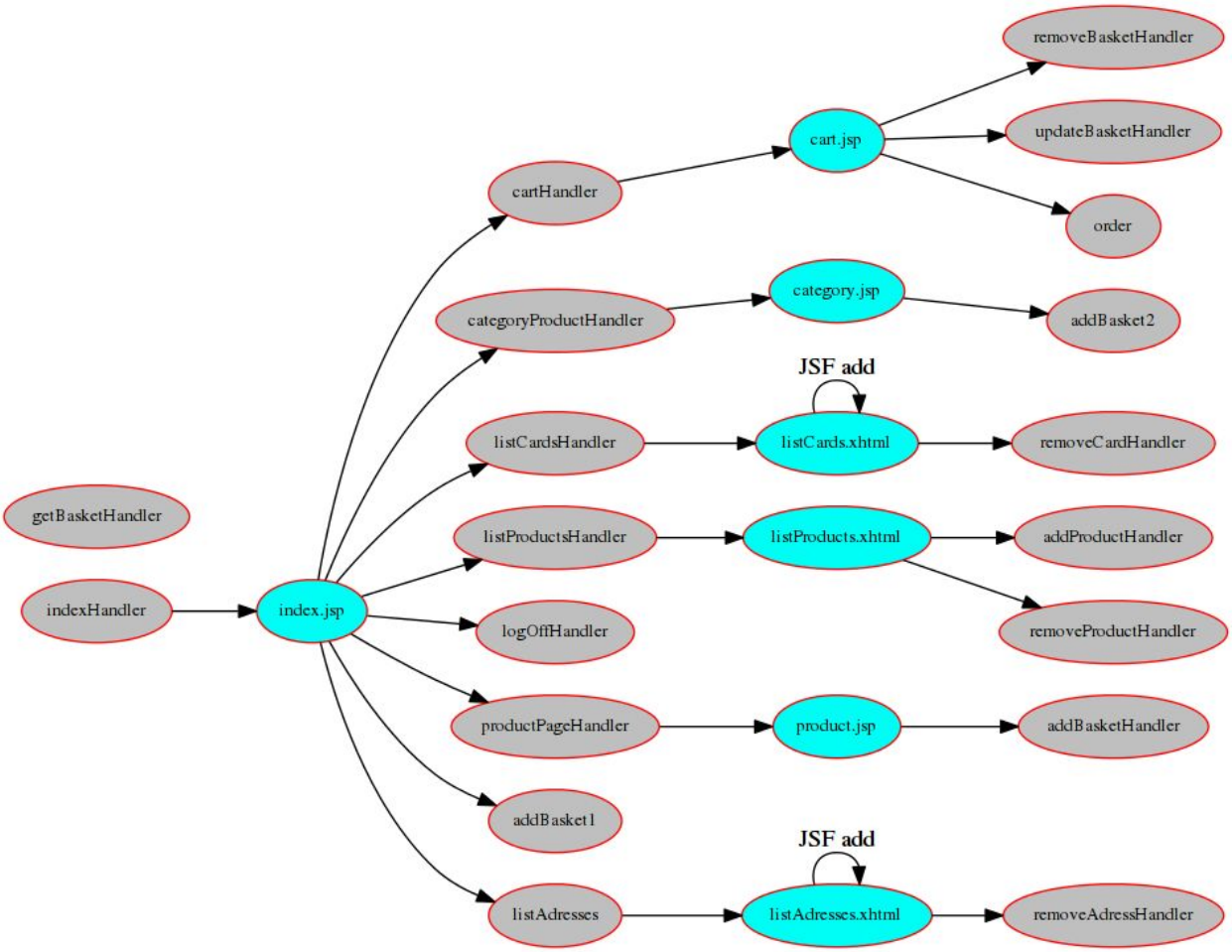
Utilisateur connecté :



Utilisateur non connecté



Entreprise


removeBasketHandler
cart.jsp
updateBasketHandler
cartHandler
order
category.jsp
categoryProductHandler
addBasket2
JSF add
listCardsHandler
listCards.xhtml
removeCardHandler
getBasketHandler
listProductsHandler
listProducts.xhtml
addProductHandler
indexHandler
index.jsp
logOffHandler
removeProductHandler
productPageHandler
product.jsp
addBasketHandler
addBasket1
JSF add
listAdresses
listAdresses.xhtml
removeAdressHandler

Description de l’architecture

Afin de respecter le modèle MVC 2 ainsi que la structure d’une application entreprise, notre projet se découpe en plusieurs parties de la façon suivante :

- Une unique servlet qui fait office de Contrôleur pour l’ensemble de l’application
- Des Handlers
- Des Entités
- Des Contrôleurs
- Des SessionBeans
- JPA pour interagir avec la BDD
- Des JSP, JSF afin de générer des Views

Notre web.xml redirige toutes les requêtes avec l’URL qui commencent par /web/* vers le contrôleur qui possède une HashMap des différents Handler en fonction de cette URL.

Il redirige aussi toutes les requêtes dont les URL commencent par /faces/* vers la servlet JSF.

Les Handler possèdent tous la méthode execute qui prend en paramètre request et response qui effectue son travail puis renvoie soit une URL vers une page JSP, soit une URL qui sera gérée par le Contrôleur.

Les Entités représentent les différentes tables de la base de donnée sous forme d’objet.

Les contrôleurs permettent de gérer la base de donnée à l’aide des Entités.

Les SessionBeans sont utilisés par les pages JSF.

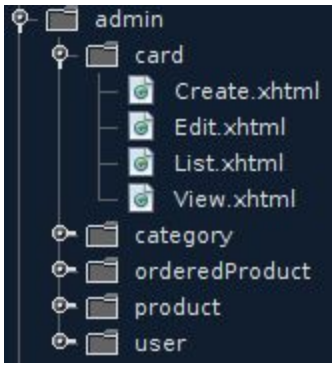
Les JSP ainsi que les JSF sont des fichiers contenant essentiellement du HTML avec du JSTL et des EL pour avoir un affichage dynamique des informations stockées dans la BDD.

Description de la partie admin

La partie admin n'est accessible uniquement par l'administrateur. Pour cela nous avons créé un filtre qui s'applique à la servlet JSF comme suit :

```
@Override
public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
    HttpServletRequest req =  (HttpServletRequest) request;
    HttpSession session = req.getSession();
    User u = (User) session.getAttribute("user");
    if(u==null || (u.getType()!=null && !u.getType().equals("admin"))) {
        ((HttpServletResponse) response).sendRedirect(req.getContextPath()+"/web/index");
    } else {
      chain.doFilter(request, response);
    }

}
```

Les pages JSF présentes dans le dossier admin sont organisés comme suit :



Chaque sous dossier représente un entité de la base de donnée.
Chaque sous dossier content 4 pages JSF correspondant à :

- Création d'une nouvelle entité
- Édition d'une entité
- Liste des entités
- La vue d'une entité

Description de la couche métier

```java
public class GetBasketHandler extends Handler{

    @Override
    public String execute(HttpServletRequest request, HttpServletResponse response) {
        HttpSession session = request.getSession();
        String lang = (String) session.getAttribute("javax.servlet.jsp.jstl.fmt.locale.session");
        User u = (User) request.getSession().getAttribute("user");
        List<OrderedProduct> l= orderedProductFacade.findCustomerId(u.getId());
        Integer num = 0;
        double total = 0;
        for(OrderedProduct bb:l){
            Product p = bb.getProduct();
            if(p==null){
                int i=0;
            }else if(p.getPrice()==null){
                int j=0;
            }
            double d = p.getPrice().doubleValue();
            int i = bb.getQuantity();
            total+=d*i;
            num+=i;
        }
        if(lang.equals("en")) total = round(total*0.918906501,2);
        else total=round(total,2);
        try {
            response.setContentType("application/json");
            PrintWriter out = response.getWriter();
            JSONObject json = new JSONObject();
            json.put("success", new Boolean(true));
            json.put("total", total);
            json.put("item", num);
            json.put("lang", lang);
            System.out.println(json);
            json.writeJSONString(out);
        } catch (IOException ex) {
            Logger.getLogger(AddBasketHandler.class.getName()).log(Level.SEVERE, null, ex);
        }
        return "";
    }

    private double round(double value, int places) {
        BigDecimal bd = new BigDecimal(value);
        bd = bd.setScale(places, RoundingMode.HALF_UP);
        return bd.doubleValue();
    }
}
```

La couche métier est représentée par nos handlers comme le montre l'exemple ci-dessus.
L'exemple présent représente l'ajout d'un produit dans le panier.
En effet, si le produit n'est pas déjà présent dans le panier, un javabean est créé puis insérer dans la base de donnée à l'aide du contrôleur de produit.
Si le produit y existe déjà, nous le récupérons, incrémentons sa quantité de 1 puis modifions l'élément en base de donnée.
La suite est utile au jQuery que nous expliquerons plus tard.

Base de donnée :

Nous avons décidé de faire 8 tables différentes.

La table USER est utilisée afin de répertorier tous les clients ainsi que les entreprises qui mettent leurs produit.
Elle est constituée de :

- D'un id afin de différencier les différents utilisateurs.
- D'un nom d'utilisateur
- D'un mot de passe
- Du prénom
- Du nom
- Du type (client normal, entreprise ou admin)
- Du mail
- D'un téléphone

La table Adresse, utilisée afin de stocker les adresses des clients. Elle est composée de:

- Un id
- Le numéro plus la rue
- La ville
- Le pays
- L'id de l'utilisateur

La table CARD, utilisée afin de stocker les cartes bleus des clients. Elle contient :

- Un id
- Le numéro de la carte
- De l'id du client

La table CATEGORY contenant toutes les catégories de produits différents.
Elle contient :

- Un id
- Un id de la catégorie supérieure afin de pouvoir faire des sous-catégories
- Un nom en français
- Un nom en anglais

La table PRODUCT qui représente les produits présents sur le site, elle est constituée de :

- D'un id
- D'un nom
- D'une description
- D'un prix

- Du chemin parvenant à l'image
- De l'id de l'entreprise qui vend ce produit
- De l'id de la catégorie à laquelle le produit est associé
- De la date de dernière modification

La table ORDEREDPRODUCT qui va représenter le panier du clien qui est composé de :

- L'id de l'utilisateur
- L'id du produit
- La quantité

la table COMMANDE qui représente les différentes commandes déjà éffectuées :

- Un id
- L'id de l'utilisateur
- Le prix total
- La date de la commande
- L'id de la carte
- L'id de l'adresse

La table CommandedProduct qui représente les produits déjà commandés.

- L'id de la commande
- L'id du produit
- La quantité

Configuration de JPA

Création d'une pool de connection dans le fichier sun-ressources.xml comportant toutes les données importantes pour se connecter à la base de donnée comme l'URL, l'username et le mot de passe.

Utilisation de cette pool de connection via le fichier persistance.xml dans lequel nous avons demandé d'afficher les différentes requêtes effectués sur la base de donnée.

Module EJB

Le module EJB ne contient que les beans entités et les bean controlleurs.
Les beans entités ont été automatiquement générés via la base de donnée.
Certaines NamedQuery ont été rajoutées afin de pouvoir intéragir avec la base de donnée d'une certaines façon.
Les beans contrôleurs ont eux aussi été générés par netbeans. Cependant certaines fonctions étant rajoutées en fonctions des besoins de l'application comme par exemple récupérer une liste de produit présent dans une certaine catégorie ou appartenant à une certaine entreprise.

Code AJAX

Afin d'avoir un meilleur rendu, nous avons fait de l'AJAX via l'API AJAX de jQuery afin de rendre les accès à la base de données de façon asynchrone à la navigation sur la page.

De ce fait nous pouvons, par exemple, ajouter plusieurs produits à notre panier sans quitter la page une seule fois.

Ce code AJAX a pour effet d'envoyer des requêtes HTML à la servlet en envoyant des données comme par exemple l'id d'un produit à ajouter.
La servlet ne fait donc ni redirection ni forward à la fin de l'exécution du Handler, elle ne fait que transmettre un objet de type JSON au code AJAX.

Lorsque le JSON est reçu par la servlet, il met à jour la valeur du panier et en fonction de la méthode qui lui est associé, il peut retirer des balises complètes comme par exemple la ligne correspondant au produit enlevé du panier lors de l'extraction de ce dernier du panier.

Bilan du projet

Nous sommes parti sur l'idée de créer un site comme Amazon, c'est à dire que l'on est qu'un intermédiaire entre le client et les entreprises. Pour réaliser ce projet, il a fallu créer une interface unifié entre les deux protagonistes qui permet aux entreprises de rajouter leurs produits et aux clients de faire leurs choix parmi tout notre contenu de façon transparente, sans connaître l'entreprise chez qui ils achètent.

Lors de la création de notre projet, les parties les plus compliqués étaient la partie Handler et Datastore. Il nous a fallu du temps pour élaborer puis implémenter la partie Handler qui permet au contrôleur Servlet d'agir sans connaître le fonctionnement exact du Handler.

L'organisation du travail a débuté par un Brainstorming sur le fonctionnement global de l'application, et surtout sur la façon de faire fonctionner le contrôleur Servlet (choix des Handler) et la partie base de données (création du datastore). Ensuite les tâches se sont répartie de la suivante, Yoann a travaillé sur le code JEE et Guillaume sur les JSP.

Toutes les techniques JEE ont été utilisées, le Java appliqué au web, les JSP avec le JSTL et les EL afin respecter le modèle MVC ainsi que les JSF. On a décidé d'utiliser en plus un framework HTML afin d'avoir une interface utilisateur plus jolie et plus interactive ainsi qu'une bibliothèque javascript afin d'accélérer notre développement de l'interface.