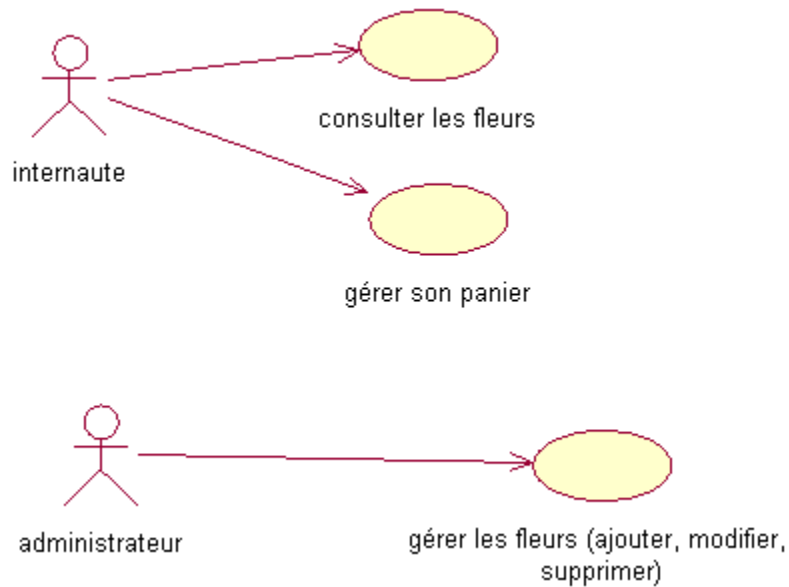


# MVC en PHP

## 1) Cas d'utilisation

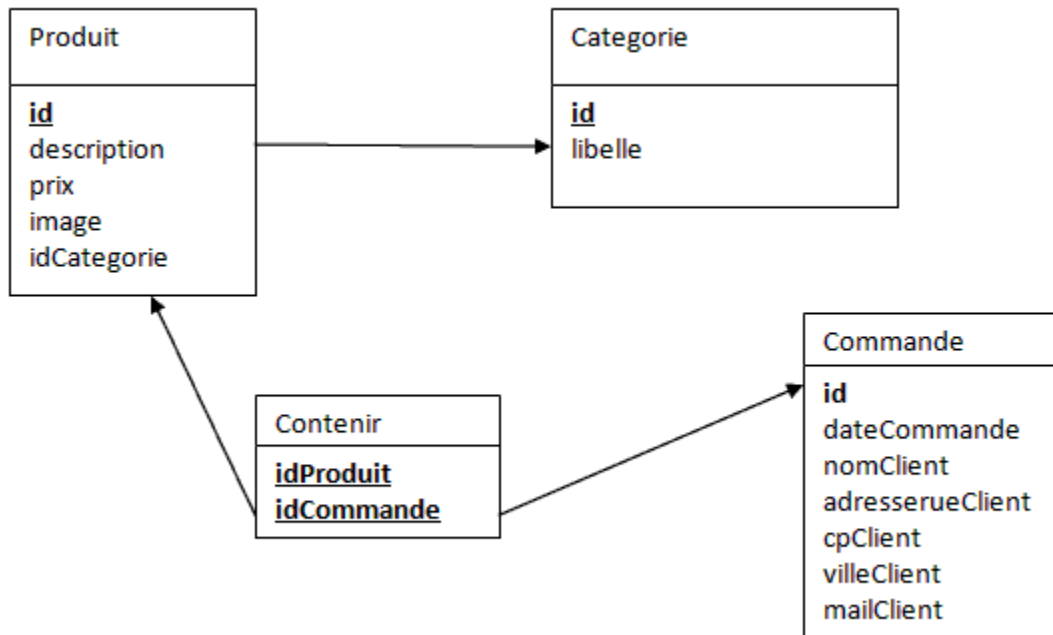
Je n'aborderai que trois cas d'utilisation, deux du *front office* (acteur l'internaute) et un du back office (acteur l'administrateur) :



Deux cas seront développés (ceux du Front Office), le dernier est donné en exercice, avec de nombreux éléments de correction.

La définition des cas d'utilisation est une étape très importante, c'est à partir de ce découpage que s'organisera l'application ; il ne faut absolument pas négliger cette étape.

## 2) Modèle de données



La base de données est sous MySql,

**Voir le script1 de la base (tables et occurrences)**

## 3) Le modèle MVC

Ce modèle de développement distingue 3 fonctionnalités :

**3.a La vue (V)** représente ce qui est exposé à l'utilisateur, en général il s'agit de HTML statique ou généré par du php ; il y a deux sortes de vue :

- + Les pages d'information navigables grâce à des liens
- + Les formulaires de saisies d'informations ; ces formulaires peuvent être présentées à plusieurs reprises pour confirmation ou signalement d'erreurs.

Une vue propose une partie commune à chaque page et des zones liées à la demande de l'utilisateur :

Bandeau	
Menu	Contenu

En général le bandeau est du code statique et les zones Menu et Contenu sont générées dynamiquement. Ainsi, chaque vue contiendra des *include* sur les fichiers correspondants ; on peut envisager que dans certaines vues les parties Menu et Contenu soient dans un même fichier car logiquement liées (ce sera le cas dans notre développement).

La mise en forme (disposition) est gérée grâce aux balises div et une feuille de style.

Ainsi une vue peut être construite avec plusieurs *sous-vues*, chaque sous-vue est un fichier php contenant presque exclusivement du html (généré dynamiquement ou non).

Par la suite je distinguerai la notion de **vue externe** de la notion de *vue* :

- La **vue externe** est la page que voit l'internaute
- Cette *vue externe* est construite à partir de vues (sous-vues ou zones) : chaque vue (sous-vues) est dans un fichier distinct.

Remarque : par ailleurs dans notre application *Lafleur*, la vue externe contiendra au moins 5 vues :

- Afin de distinguer l'en-tête (partie *head, meta*) du bandeau, il y aura une vue en-tête.
- Une vue bandeau
- Une vue menu
- Une vue contenu
- Une vue pied (balises fermantes HTML)

En-tête	
Bandeau	
Menu	Contenu
Pied	

## 3.b Le contrôleur

Ce sont les contrôleurs qui vont être à l'écoute des requêtes de l'utilisateur et fournir ainsi la *vue externe* correspondante (constituée de sous-vues). Pour cela, il faudra à tout moment connaître **l'état de l'application** c'est à dire le contexte de la demande : *"la page demandée fait suite à quelle **action** précise de l'utilisateur ?"* C'est au contrôleur de connaître l'état applicatif en testant une variable qui sera nommée **\$action**, *provenant d'une requête POST ou GET*. Ainsi le contrôleur se présentera souvent sous le format suivant :

```
$action = $_REQUEST['action'];
switch($action)
{
case 'ceci' :
include("vues/v_ceci.php");
include("vues/v_encorececi.php");
case 'cela' :
include("vues/v_cela.php");
}
```

Je ne distinguerais pas les variables POST ou GET, elles sont présentes dans le tableau **\$\_REQUEST**.

On pourra trouver qu'il y a parfois redondance d'instructions dans les contrôleurs ; ceci est justifié par un désir de clarté dans la présentation des responsabilités des options des contrôleurs.

On trouvera un fichier (préfixé par c\_) contrôleur par cas d'utilisation ainsi qu'un *contrôleur principal* qui oriente vers les différents contrôleurs.

## 3.b Le modèle

C'est la couche (bibliothèque de fonctions ou de classes) qui accède à la base

de données. Ici nous avons utilisé une classe PDO, dédiée à l'accès aux données ; cette classe -présente par défaut dans PHP- a l'avantage d'être générique (indépendante du SGBD, c'est ce que l'on appelle l'abstraction d'accès aux données-PDO, entity framework, ADO, JDBC, ODBC sont de outils d'abstraction qui permettent de coder et de ne pas s'occuper du sgbd (mysql,MSsql, Oracle,..) et fournit des services avancés, comme celui de faire à "notre" place les boucles de parcours des jeux d'enregistrements. Par exemple, pour récupérer toutes les catégories de fleurs, il suffit de faire :

```
/**
 * Retourne toutes les catégories sous forme d'un tableau associatif
 *
 * @return le tableau associatif des catégories
 */
public function getLesCategories()
{
    $req = "select * from categorie";
    $res = PdoLafleur::$monPdo->query($req);
    $lesLignes = $res->fetchAll();
    return $lesLignes;
}
```

Cette fonction **retourne un tableau** à deux dimensions.

La vue qui va utiliser cette fonction va parcourir la tableau et insérer du code html

```
:
<ul id="categories">
<?php
    foreach( $lesCategories as $uneCategorie)
    {
        $idCategorie = $uneCategorie['id'];
        $libCategorie = $uneCategorie['libelle'];
        ?>
        <li>
            <a href=index.php?uc=voirProduits&categorie=<?php echo $idCategorie ?>&action=voirProduits><?php echo $libCategorie ?></a>
        </li>
    }
<?php
?>
</ul>
```

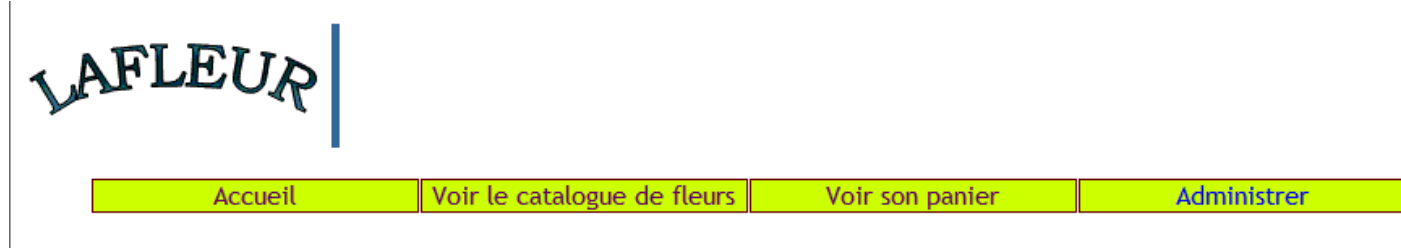
Cette technologie **charge en mémoire un tableau de données** (couche modèle), le contrôleur va fournir ce tableau à la vue qui parcourt le tableau pour placer la couche de présentation (en html). Ce fonctionnement a bien sûr un coût en terme d'utilisation de ressources machine ; c'est le prix à payer pour un développement plus cohérent. Par contre ceci diminue les temps de connexion à la base de données. Les framework (zend, Symfony) qui mettent en oeuvre cette techno de manière plus industrielle utilisent des mécanismes de cache (sur disque) ou de chargements partiels paramétrables.

Dans la couche modèle peuvent figurer aussi des fonctions *métiers*, règles de

gestion du contexte.

## 4) Mise en oeuvre dans l'application Lafleur.

La page index joue le rôle de contrôleur principal ; c'est à lui de dispatcher vers les trois cas d'utilisation. La vue externe est :



On retrouve les 3 cas d'utilisation.

```
<?php
session_start();
include("vues/v_entete.php") ;
include("vues/v_bandeau.php") ;
if(!isset($_REQUEST['uc']))
    $uc = 'accueil';
else
    $uc = $_REQUEST['uc'];
switch($uc)
{
    case 'accueil':
        {include("vues/v_accueil.php");break;}
    case 'voirProduits' :
        {include("c_voirProduits.php");break;}
    case 'gererPanier' :
        { include("c_gestionPanier.php");break; }
    case 'administrer' :
        { include("c_gestionProduits.php");break; }
}
include("vues/v_pied.php") ;
?>
```

*code de index.php*

Ainsi le **système ne fournit qu'une seule page**, la page *index.php* dans laquelle sont inclus différents fichiers php.

### 4.a ) Cas d'utilisation *voirProduits*

Ce cas propose les deux vues externes suivantes :

Après l'appel	Après le choix de la catégorie
---------------	--------------------------------

du catalogue	
--------------	--

Fleurs
Plantes
Composition



COMORES PASTEL

Bouquet de roses  
multicolores

: 58.00 Euros



GRENADINES

Bouquet de roses rouges

: 50.00 Euros



En fait ce cas d'utilisation propose 3 *états* ; les deux premiers ci-dessus et un troisième correspondant au rechargement de cette page après ajout d'un produit au panier; c'est ainsi que le contrôleur présente 3 *actions* :

```

$action = $_REQUEST['action'];
switch($action)
{
    case 'voirCategories':
    {
        $lesCategories = $pdo->getLesCategories();
        include("vues/v_categories.php");
        break;
    }
    case 'voirProduits' :
    {
        $lesCategories = $pdo->getLesCategories();
        include("vues/v_categories.php");
        $categorie = $_REQUEST['categorie'];
        $lesProduits = $pdo->getLesProduitsDeCategorie($categorie);
        include("vues/v_produits.php");
        break;
    }
    case 'ajouterAuPanier' :
    {
        $idProduit=$_REQUEST['produit'];
        $categorie = $_REQUEST['categorie'];
        $ok = ajouterAuPanier($idProduit);
        if(!$ok)
        {
            $message = "Cet article est déjà dans le panier !!";
            include("vues/v_message.php");
        }
        $lesCategories = $pdo->getLesCategories();
        include("vues/v_categories.php");
        $lesProduits = $pdo->getLesProduitsDeCategorie($categorie);
        include("vues/v_produits.php");
        break;
    }
}

```

C'est la valeur de la variable `$_REQUEST['action']` qui permet d'aiguiller vers tel ou tel case.

C'est pourquoi il est fondamental de suivre l'**état applicatif** en *passant de page en page* une variable *action*. Ainsi la vue correspondant au choix *voirCategories* n'est visible que si un lien est construit avec la bonne valeur de la variable *action* :

```

<ul id="menu">
  <li><a href="index.php?uc=accueil"> Accueil </a></li>
  <li><a href="index.php?uc=voirProduits&action=voirCategories"> Voir le catalogue d

```



*extrait du bandeau de la page d'accueil*

Le paramètre **uc** fournit le cas d'utilisation, le paramètre **action** indique l'étape dans le cas.

De même pour le choix de visualisation des produits après sélection de la catégorie :

```
<?php
foreach( $lesCategories as $uneCategorie)
{
    $idCategorie = $uneCategorie['id'];
    $libCategorie = $uneCategorie['libelle'];
    ?>
    <li>
        <a href=index.php?uc=voirProduits&categorie=?php echo $idCategorie ?>&action=voirProduits><?php echo $libCategorie ?></a>
    </li>
}
?>
```

#### **4.b ) Cas d'utilisation *gererPanier***

Le principe est le même ; les *vues externes* associées sont :

- Première visualisation du panier, on vient de l'accueil

# Votre panier



COMORES PASTEL

Bouquet de roses multicolores(58.00 Euros)



SAINTE MARIE JAUNE

Bouquet de roses jaunes(78.00 Euros)



- On rappelle le formulaire après avoir supprimé un article :

## Votre panier



SAINTE MARIE JAUNE

Bouquet de roses jaunes(78.00 Euros)



- On demande à passer commande :

Commande

Nom et prenom*	<input type="text"/>
rue*	<input type="text"/>
code postal*:	<input type="text"/>
ville*:	<input type="text"/>
mail*:	<input type="text"/>

Valider	Annuler
---------	---------

- Le formulaire est retourné (avec les erreurs éventuelles)

- erreur de code postal
- erreur de mail

Commande

Nom Prénom*	<input type="text" value="toto"/>
rue*	<input type="text" value="12 rue Petit"/>
code postal*	<input type="text" value="7501"/>
ville*	<input type="text" value="paris"/>
mail*	<input type="text" value="toto.fr"/>

Le contrôleur doit répondre à ces 4 états :

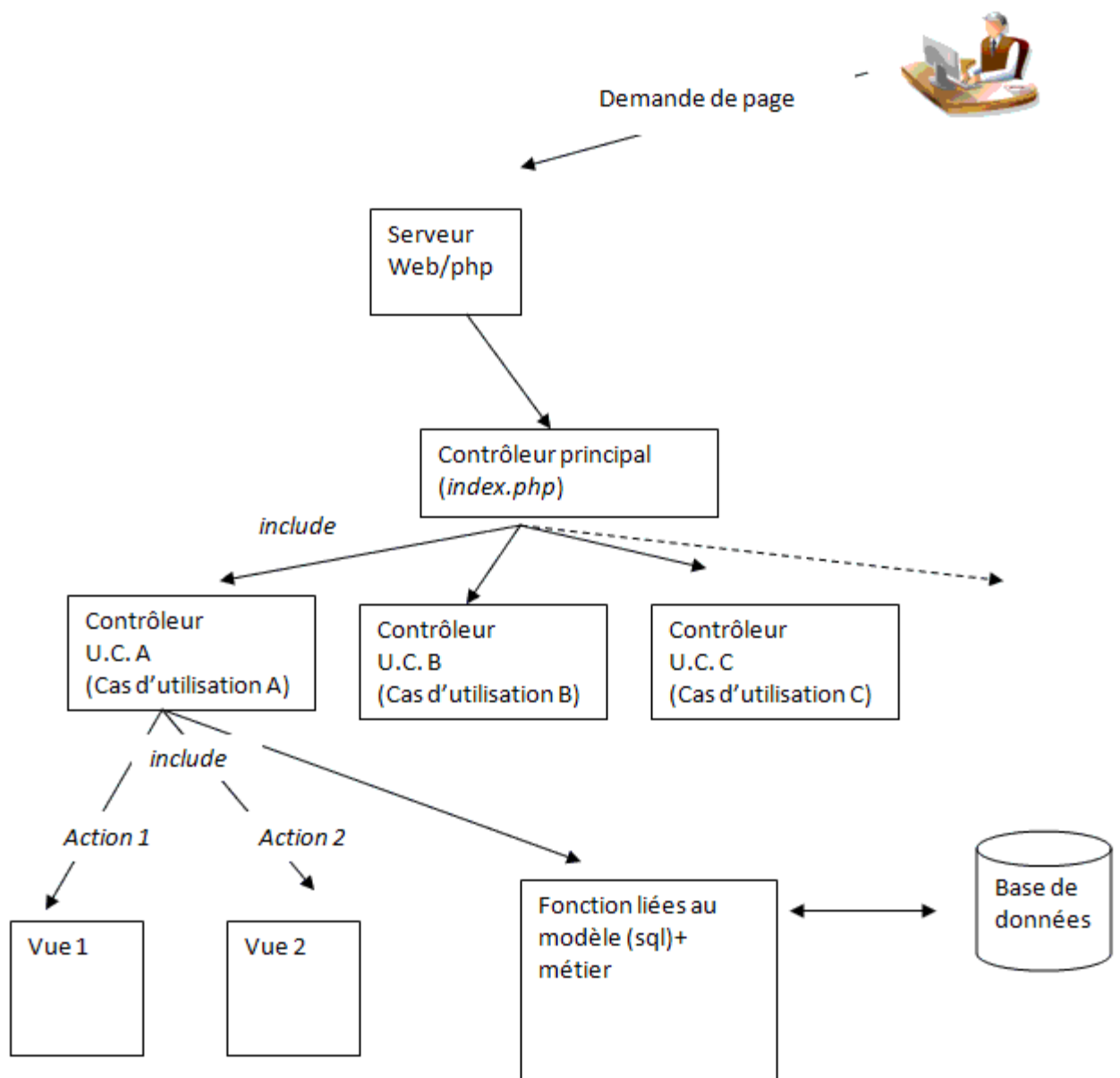
```

$action = $_REQUEST['action'];
switch($action)
[
{
    case 'voirPanier':
[
    {
        $n= nbProduitsDuPanier();
        if($n >0)
[
        {
            $desIdProduit = getLesIdProduitsDuPanier();
            $lesProduitsDuPanier = $pdo->getLesProduitsDuTableau($desIdProduit);
            include("vues/v_panier.php");
-
        }
        else
[
        {
            $message = "panier vide !!";
            include ("vues/v_message.php");
-
        }
        break;
-
    }
    case 'supprimerUnProduit':
[
    {
-
    case 'passerCommande' :
[
    {
-
    case 'confirmerCommande' :
[
    {
-
}
-
}

```

Remarque : on a "plié" une partie du code afin de montrer l'essentiel, les 4 case correspondant aux 4 actions.

On peut schématiser le fonctionnement du chargement des pages (include) :



**L'utilisateur ne chargera que la page index dans laquelle seront inclus les contrôleurs et vues correspondant à l'action demandée.**

#### **4.c Gestion des erreurs**

Les erreurs liées à des saisies incorrectes sont signalées dans une sous-vue distincte (*v\_erreurs.php*) ; cette vue est incluse ou pas à la fin du formulaire de saisie reposté :

---

```

<div class="erreur">
<ul>
<?php
    foreach($msgErreurs as $erreur)
    {
        ?>
        <li><?php echo $erreur ?></li>
    }
<?php
?>
</ul>
</div>
|

```

code du fichier *v\_erreurs.php*

Ce fichier est inclu si une erreur est décelée :

---

```

case 'confirmerCommande' :
{
    $nom=$_REQUEST['nom'];$rue=$_REQUEST['rue'];$ville=$_REQUEST['ville'];$cp=
    $msgErreurs = getErreursSaisieCommande($nom,$rue,$ville,$cp,$mail);
    if (count($msgErreurs) !=0)
    {
        include ("vues/v_erreurs.php");
        include ("vues/v_commande.php");
    }
    else
    {

```

La fonction *getErreursSaisieCommande* retourne un tableau de messages d'erreurs :






```

function getErreursSaisieCommande($nom,$rue,$ville,$cp,$mail)
{
    $lesErreurs = array();
    if($nom=="")
    {
        $lesErreurs[]="Il faut saisir le champ nom";
    }
    if($rue=="")
    {
        $lesErreurs[]="Il faut saisir le champ rue";
    }
    if($ville=="")
    {
        if($cp=="")
        {
            else
            {
                if($mail=="")
                {
                    else
                    {
                        return $lesErreurs;
                    }
                }
            }
        }
    }
}
?>

```

## 4.d Organisation des fichiers

Voici une organisation possible :

Nom	Modifié le	Type	Taille
 controleurs	29/08/2017:09:02	Dossier de fichiers	
 images	03/09/2017:13:54	Dossier de fichiers	
 util	29/08/2017:17:14	Dossier de fichiers	
 vues	03/09/2017:08:20	Dossier de fichiers	
 index	29/08/2017:19:15	Fichier PHP	

Les contrôleurs (c\_...) sont dans le répertoire controleur, les vues (v\_...) et sous-



vues sont dans un répertoire particulier, le répertoire util contient les fichiers du modèle ainsi que le fichier css.

## 4.e Gestion du style CSS

Un fichier central est utilisé pour toute la partie présentation ; l'application se conforme aux recommandations du W3C concernant les feuilles de style.

# 5) Démarche conseillée

## 5.1 Définir les cas d'utilisation.

D'abord en décomposant l'application en petites fonctionnalités (gérer un produit, éditer un état, passer une commande, etc...). Trouver les acteurs, dessiner le diagramme des cas d'utilisation.

## 5.2 Préciser chaque cas d'utilisation

Indiquer textuellement les interactions entre l'utilisateur et le système ; par exemple le premier cas d'utilisation (consultation des produits ) pourrait se présenter ainsi :

Nom du cas : consulter les articles (fleurs)

Acteur : un internaute

Scénario normal (le plus fréquent)

1. **L'internaute demande à consulter les articles**

2. Le système retourne la liste des catégories

3. **L'internaute sélectionne une catégorie**

4. Le système retourne la liste des articles de la catégorie choisie

Scénario étendu

5. **L'internaute demande à déposer un article dans son panier**

6. le système ajoute l'article au panier

Scénario particulier

6.1 L'article figure déjà dans le panier : le système en informe l'utilisateur

On remarque que l'on retrouvera bien les trois *actions* (en gras) présentes dans le contrôleur du cas d'utilisation (cf plus haut)

De même pour la gestion du panier

Nom du cas : gérer le panier

Acteur : l'internaute

Scénario normal

1. **L'internaute demande à voir son panier**

2. Le système retourne la liste des articles du panier

Scénario étendu

3. **L'internaute demande la suppression d'un article**
4. Le système retire l'article du panier (après confirmation)
5. **L'internaute demande à passer commande**
6. Le système retourne un formulaire pour saisies
7. **L'internaute remplit le formulaire et l'envoie**
8. Le système enregistre la commande

Scénario particulier

2.1 Le panier est vide : le système en informe l'utilisateur

8.1 Le système constate une erreur de saisie ; il en informe l'internaute, retour à 6

Nous retrouvons les 4 actions et la gestion d'erreur alternative dans le contrôleur présenté plus haut.

Cette phase de description textuelle est donc très importante.

### **5.3 Imaginer les vues externes**

Il s'agit ici de dessiner comment se présenteront chaque page, telle que la voie l'internaute. Ceci permet de mettre en évidence les sous-vues qui peuvent être partagées par plusieurs cas d'utilisation, cf plus haut pour le bandeau principal ou l'affichage de la sous-vue des erreurs.

### **5.4 Construire l'architecture physique de l'application.**

Créer les répertoires, les contrôleurs vides de code, le contrôleur principal, etc...

### **5.5 Ecrire le contrôleur principal**

C'est le fichier index.php avec le switch pointant sur chaque cas d'utilisation

### **5.6 Coder chaque cas d'utilisation**

et tester.

## **Travail à faire**

*Voir le code lafleur.zip*

### **Exercice 1**

**Le cas d'utilisation de Back Office (gestion des produits) n'est pas traité. Développer ce cas en s'appuyant sur le cas d'utilisation présenté plus bas et en suivant la démarche proposée.**

Nom du cas : gérer les articles (fleurs)

Acteur : l'administrateur

Scénario normal (le plus fréquent)

1. L'administrateur demande à se connecter
2. Le système demande le nom de user et le mot de passe.
3. L'internaute saisit le nom de user et le mot de passe
4. Le système retourne la liste des catégories
5. L'administrateur sélectionne une catégorie
6. Le système retourne la liste des articles de la catégorie (id et description) avec pour chaque article la possibilité de modifier ou supprimer. Une option d'ajout de produit pour cette catégorie est proposée
7. L'administrateur sélectionne modifier
8. Le système retourne les infos de l'article, description et prix.
9. L'administrateur modifie les infos et envoie le formulaire
10. le système enregistre la modification

Extensions

- 11 L'administrateur sélectionne supprimer
- 12 Le système demande si l'article doit bien être supprimé
- 13 L'administrateur confirme ou pas
- 14 Le système supprime ou non (selon la réponse en 7.3)
- 15 L'administrateur demande à créer un nouveau produit
- 16 Le système retourne un formulaire de saisies
- 17 L'administrateur remplit le formulaire et envoie le formulaire
- 18 Le système enregistre les informations
- 19 A tout moment l'administrateur demande à se déconnecter
- 19.1 Le système le déconnecte et retourne la page d'accueil

Scénarios particuliers

- 4.1 Les infos de connexion sont invalides. Retour à 2 (avec un message d'erreur)
- 10.1 Les infos reçues ne sont pas valides. Retour à 8 (avec un message d'erreur)
- 18.1 des informations ne sont pas valides ; retour à 15 (avec un message d'erreur)
- x.1 A tout moment, l'administrateur demande à retourner dans l'administration
- x.2 Le système ne lui retourne pas de formulaire de connexion mais le catalogue des catégories.

- Dessiner rapidement les vues externes.
- Coder les pages

Remarque : une table **Administrateur** a été créée avec deux lignes :

```
INSERT INTO Administrateur VALUES ('1','toto','toto');  
INSERT INTO Administrateur VALUES ('2','titi','titi');
```

Voir les éléments de corrigé exo 2

## Exercice 2

On désire modifier une règle de gestion importante. Actuellement, l'internaute ne peut mettre dans son panier qu'une seule occurrence d'un article. On souhaite pouvoir offrir la possibilité de mettre plusieurs occurrences d'un article.

Pour cela, il faudra changer la base de données.

1.1) Modifier la base de données. Ceci se fera avec un script qui modifiera la structure d'une table.

1.2) On vous fournit le cas d'utilisation "gestion du panier" modifié par cette nouvelle règle de gestion :

Nom du cas : gérer le panier

Acteur : l'internaute

Scénario normal

**1. L'internaute demande à voir son panier**

2. Le système retourne la liste des articles du panier *avec par défaut le nombre à 1 pour chacun*

Scénario étendu

**3. L'internaute demande la suppression d'un article**

4. Le système retire l'article du panier (après confirmation)

**5. *L'internaute modifie les quantités et demande à passer commande***

6. Le système retourne un formulaire pour saisies

**7. L'internaute remplit le formulaire et l'envoie**

8. Le système enregistre la commande

Scénario particulier

2.1 Le panier est vide : le système en informe l'utilisateur

8.1 Le système constate une erreur de saisie ; il en informe l'internaute, retour à 6

**Nous avons mis en italique les ajouts au cas initial.**

Apporter les modifications nécessaires à l'application