

Mission 2 - GsbParam et Git

L'application GsbParam est une application PHP qui gère un panier électronique de produits de parapharmacie.

Cette application a été développée avec la technologie MVC (Modèle Vue Contrôleur). Elle va vous permettre de vous perfectionner avec cette architecture.

Un dossier GsbParam contenant le code de l'application vous est fourni ainsi que le script de création de la base de données MariaDB qui contient les données relatives à l'application dans le dossier BD de GsbParam.

Afin de vous donner une vision globale du comportement fonctionnel de l'application, le paragraphe suivant vous présente le diagramme de cas d'utilisation (qui fait partie des diagrammes de l'analyse UML). Le rôle de ce diagramme de cas d'utilisation est de présenter de manière simple ce que fait l'application et quels sont ses acteurs (utilisateurs).

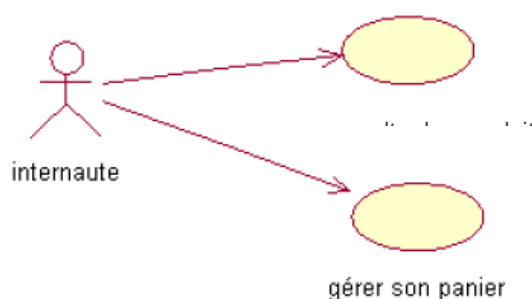
Ensuite la structure de l'application vous sera présentée et son fonctionnement expliqué. Au fur et à mesure de sa lecture, rédigez un compte rendu pour répondre aux questions posées.

I. Cas d'utilisation

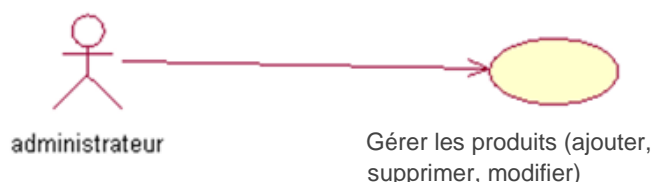
La définition des cas d'utilisation est une étape importante, c'est à partir de ce découpage que s'organisera l'application.

Les trois cas d'utilisation schématisés ci-dessous par des ovales deviendront des 'case' à traiter par le contrôleur de l'application (en MVC, c = Contrôleur).

Parmi les trois cas d'utilisation, deux concernent le *front office* (acteur= l'internaute) et l'autre, le *back office* (acteur= l'administrateur) :

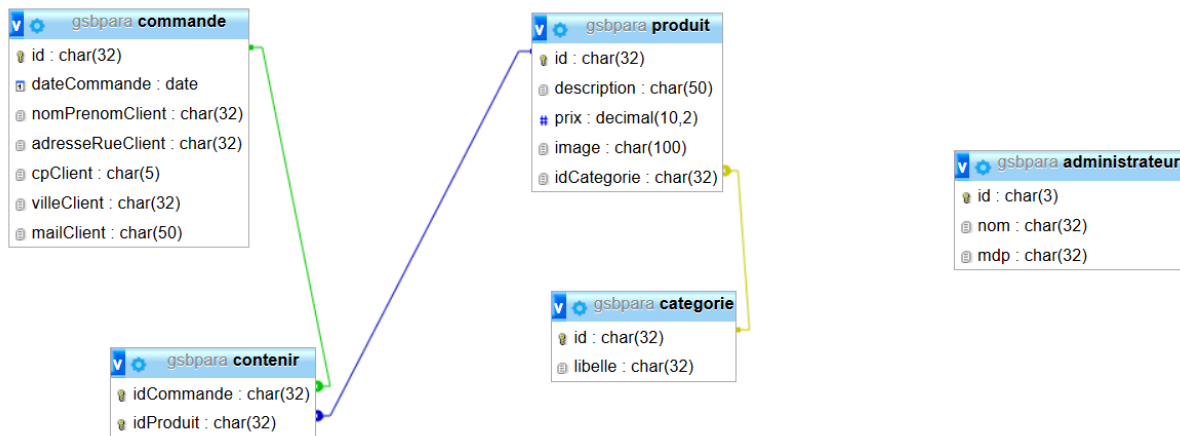


Les deux cas ci-dessus (consulter les produits, et gérer son panier) sont développés (ils vous sont fournis dans le code de l'application). Le cas d'utilisation ci-dessous sera à développer.



II. Modèle de données

1. Structure de la base de données (modèle physique)



La base de données est une base de données MariaDB,

Le script de la base (tables et occurrences) est fourni dans le dossier BD du dossier GsbParam.

La table administrateur n'est pas reliée aux autres car elle contiendra uniquement les données pour une connexion en mode administrateur (donc indépendante de la gestion des commandes).

2 Remarques

☞ Quelles remarques pouvez-vous formuler sur la table commande ?

III. Le modèle MVC

Ce modèle de développement structure les fichiers de l'application en 3 parties : la partie Vues, la partie Modeles et la partie Controleurs. Au niveau de l'arborescence de l'application, chaque partie sera stockée dans un dossier spécifique respectivement vues, modele et controleurs :

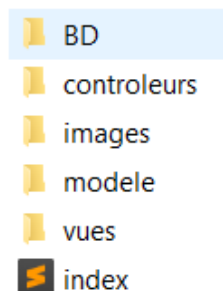


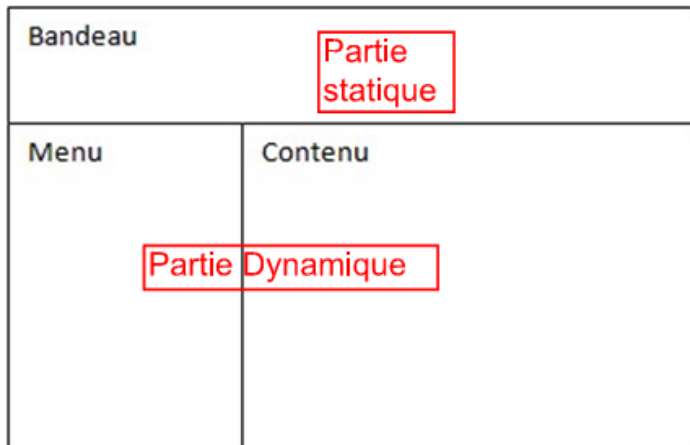
Figure 1 Arborescence de l'application

1. La vue (V)

Elle représente ce qui est exposé à l'utilisateur, en général il s'agit de code HTML statique ou dynamique (généré par du php) ; il y a deux sortes de vues :

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

Une **vue** se décompose en des zones communes à chaque page (le bandeau et le menu) et des zones liées à la demande de l'utilisateur (contenu).



Le bandeau est constitué de code statique (code HTML) et les zones Menu et Contenu sont générés dynamiquement (code PHP qui exploite les données de la BD). La mise en forme (disposition) est gérée grâce aux balises div et une feuille de style (stockée dans le dossier modele).

2. Le contrôleur (C)

Ce sont les contrôleurs qui vont être à l'écoute des demandes de l'utilisateur et qui vont fournir la *vue externe* correspondante à la demande en sollicitant des fonctions du modèle (qui exploitent les données stockées en base de données).

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 cet état applicatif en testant une variable **\$action**, provenant d'une requête POST (ou GET). Ainsi le code du contrôleur se présentera souvent sous le format suivant :

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



Rappel de la structure type d'un contrôleur

On ne distinguera pas les variables POST ou GET, toutes stockées dans le tableau **\$_REQUEST**.

Il y a parfois redondance d'instructions dans les contrôleurs ; ceci est justifié par un désir de clarté dans la présentation des responsabilités de chaque option des contrôleurs.

On trouvera **un fichier contrôleur (préfixé par c_)** pour chaque cas d'utilisation ainsi qu'un *contrôleur principal (index.php)* qui oriente vers les différents contrôleurs.

3 Le modèle

Il contient le code qui accède directement à la base de données. Ici nous avons utilisé la classe Objet nommée PDO. C'est une classe **native** de PHP dédiée à l'accès aux données ; elle a l'avantage d'être générique (indépendante du SGBD) et fournit des services avancés, comme celui de retourner des résultats sous forme de tableaux indexés ou d'objet (suivant la méthode de la classe utilisée) : `fetch()` sans paramètre retourne un tableau indexé par le nom et le numéro de la colonne.

IV. Mise en œuvre dans l'application GsbParam.

Partie Modèle : stockée dans le dossier nommé Modele.

Partie Vue : stockée dans le dossier Vues.

Partie Contrôleur : stockée dans le dossier Controleurs

1.a. Le modèle

Il contient la fonction de connexion à la base de données (dans le fichier `bd.inc.php`) qui retourne un objet de la classe `Pdo` qui servira à effectuer les requêtes dans les fonctions du fichier `bd.produits.inc.php`

```
function connexionPDO() {
// à compléter
    $login = '';
    $mdp = '';
    $bd = 'GsbPara';
    $serveur = 'localhost';

    try {
        $conn = new PDO("mysql:host=$serveur;dbname=$bd",$login,$mdp,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\''));
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        return $conn;
    } catch (PDOException $e) {
        print "Erreur de connexion PDO ";
        die();
    }
}<
```

Le login et le mdp doivent être ceux d'un utilisateur autorisé à se connecter à la BD (et à faire des opérations courantes comme SELECT, INSERT, UPDATE).

1.b. getLesCategories

Cette méthode utilise l'objet retourné par `connexionPDO()` pour récupérer et retourner toutes les catégories de produits sous la forme d'un tableau associatif à 2 dimensions ; une dimension représentant les n catégories, l'autre représentant les caractéristiques de chaque catégorie (car un `SELECT *` retourne toutes les colonnes) :

```
/**
 * Retourne toutes les catégories sous forme d'un tableau associatif
 *
 * @return array $lesLignes le tableau associatif des catégories
 */
function getLesCategories()
{
    try
    {
        $monPdo = connexionPDO();
        $req = 'select id, libelle from categorie';
```

```

$res = $monPdo->query($req);
$lesLignes = $res->fetchAll(PDO::FETCH_ASSOC);
return $lesLignes;
}
catch (PDOException $e)
{
    print "Erreur !: " . $e->getMessage();
    die();
}
}

```

Remarque : pour la sécurité des données, il est préférable d'utiliser 'select id, libelle from categorie' à 'select * from categorie' car si on ajoute un champ confidentiel à cette table, il sera récupéré par la requête, ce qui n'est pas une démarche sécuritaire.

1.c Tester getLesCategories()

☞ Copier le dossier GsbParam dans www

☞ Créer dans ce dossier, un dossier TestUnitaires qui contiendra des tests des méthodes de l'application.

☞ Créer un fichier test_getLesCategories.php.

☞ Ajouter ensuite un affichage avec var_dump() du résultat de la méthode getLesCategories().

Résultat le tableau suivant :

```

array (size=3)
  0 =>
    array (size=2)
      'id' => string 'CH' (length=2)
      'libelle' => string 'Cheveux' (length=7)
  1 =>
    array (size=2)
      'id' => string 'FO' (length=2)
      'libelle' => string 'Forme' (length=5)
  2 =>
    array (size=2)
      'id' => string 'PS' (length=2)
      'libelle' => string 'Protection Solaire' (length=18)

```

id	libelle
CH	Cheveux
FO	Forme
PS	Protection Solaire

Figure 2 - contenu de la table Categorie

On voit bien le tableau contenant 3 résultats (indices de 0 à 2). Si le tableau s'appelle \$lesCateg,, \$lesCateg[0] contient un tableau indicé sur le nom de colonne. On peut accéder au libellé de la catégorie CH par \$lesCateg[0]['libelle'].

1.d Tester getLesProduitsDeCategorie()

☞ Dans un fichier test_getLesProduitsDeCategorie.php, tester la méthode getLesProduitsDeCategorie() vérifier que les informations affichées par le var_dump sont cohérentes avec celles de la BD.

1.e. Autres fichiers du dossier Modele.

Il contient également :

- un fichier de style nommé cssGeneral.css

- un fichier fonctions.inc.php qui contient des fonctions qui n'ont pas pour but d'accéder aux données de la BD. Il contient entre autres les fonctions qui gère le panier, et les fonctions qui vérifient les informations des formulaires.

Travail à faire : à l'aide de l'outil Doxygen, réaliser la documentation des fichiers fonctions.inc.php et du fichier bd.produits.inc.php.

Par la suite, vous ajouterez des commentaires à aux nouvelles fonctions créées pour enrichir la documentation.

2 Les vues

2a vue externe

Dans *GsbParam*, la vue externe (page visible par l'utilisateur) pourra contenir jusqu'à 5 vues (stockée chacune dans un fichier différent dont le nom commence par v_) :

- Une vue *en-tête* (contient les éléments *head*, *meta* du code HTML).
- Une vue *bandeau* avec les liens 'accueil' , 'Nos produits' ...
- Une vue *menu* avec l'affichage des catégories de produits (provenant de la BD)
- Une vue *contenu* qui affichera les produits demandés
- Une vue *pied* (qui contient les balises fermantes HTML)

En-tête	v_entete.php
Bandeau	v_bandeau.php
Menu	Contenu
v_categories.php	v_produitsDeCategorie.php
Pied	v_pied.php

2.b. Un exemple de vue : v_categories.php

La vue v_categories.php doit afficher le résultat de la méthode getLesCategories() appelée par le contrôleur c_voirProduits.php grâce aux instructions suivantes

```
$lesCategories = getLesCategories();
include("vues/v_categories.php");
```

Dans le code de la vue, on trouve le parcours du tableau \$lesCategories (boucle foreach) et le code HTML pour afficher le libellé des catégories. Derrière chaque libellé, on a un lien vers index.php avec 3 paramètres *uc*, *categorie* et *action* affectés qui seront passés par l'URL.

```
<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
  ?>
```

v_categories.php

Cheveux
Forme
Protection
Solaire

Figure 3 - résultat

Le paramètre `uc=voirProduits` (passé par l'URL) est traité par le contrôleur principal **index.php** qui se charge d'inclure le code du contrôleur spécifique concerné (voir paragraphe suivant).

3. Les contrôleurs

3.a Le contrôleur principal : index.php

index.php reçoit les paramètres `uc` (qui contient le cas d'utilisation concerné et permet d'inclure un contrôleur spécifique) et `action` (qu'il fournit au contrôleur spécifique qui le traitera). Il peut recevoir aussi d'autres paramètres spécifiques au contrôleur inclus (comme `categorie` qui précise la catégorie des produits que l'on souhaite afficher).

Ce fonctionnement peut paraître lourd mais c'est le prix d'une structuration performante de l'application. Les frameworks actuels (Zend, Symfony, Silex...) utilisent cette architecture.

Voici le code du contrôleur principal index.php qui vous montre comment il active un contrôleur particulier en fonction du paramètre `uc` reçu grâce au `switch(uc)` :

```
<?php
session_start();
include("vues/v_entete.php") ;
require_once("modele/fonctions.inc.php");
require_once("modele/bd.produits.inc.php");

include("vues/v_bandeau.php") ;

if(!isset($_REQUEST['uc']))
    $uc = 'accueil'; // si $_GET['uc'] n'existe pas , $uc reçoit une
valeur par défaut
else
    $uc = $_REQUEST['uc'];

switch($uc)
{
    case 'accueil':
        {include("vues/v_accueil.php");break;}
    case 'voirProduits' :
        {include("contrôleurs/c_voirProduits.php");break;}
    case 'gererPanier' :
        { include("contrôleurs/c_gestionPanier.php");break; }
    case 'administrer' :
        { include("contrôleurs/c_gestionProduits.php");break; }
}
include("vues/v_pied.php") ;
?>
```

🔍 Regardez le code ci-dessus et indiquez quel est le contrôleur spécifique inclus par index.php quand il reçoit le paramètre `uc=voirProduits` :

3.b Un contrôleur spécifique

Le contrôleur `c_voirProduits.php` inclus, utilisera les 2 autres paramètres (`categorie` et `action`). A l'aide d'un `switch`, il teste le paramètre `action` pour déterminer les traitements à effectuer et les vues à afficher.

☞ Observez le code ci-dessous extrait de `c_voirProduits.php` et indiquez quel bloc d'instructions sera exécuté quand l'utilisateur clique sur le lien 'Forme' du menu affiché par `v_categories.php`.

```
<?php
// contrôleur qui gère l'affichage des produits
initPanier(); // se charge de réserver un emplacement mémoire pour le
panier si pas encore fait
$action = $_REQUEST['action'];
switch($action)
{
    case 'voirCategories':
    {
        $lesCategories = getLesCategories();
        include("vues/v_choixCategorie.php");
        break;
    }
    case 'voirProduits' :
    {
        $lesCategories = getLesCategories();
        include("vues/v_categories.php");
        $categorie = $_REQUEST['categorie'];
        $lesProduits = getLesProduitsDeCategorie($categorie);
        include("vues/v_produitsDeCategorie.php");
        break;
    }
    case 'ajouterAuPanier' :
```

Figure 4 - Extrait de `c_voirProduits.php`

☞ Indiquez où est utilisé le paramètre `categorie`. Quelle sera sa valeur si l'utilisateur a cliqué sur le lien `Forme` ? en get : `?uc=voirProduits&categorie=FO&action=voirProduits`

☞ Vérifiez en cliquant sur le lien `Forme` vous-même et en regardant l'URL.

3.c. Les paramètres : *uc* et *action*

Les paramètres *uc* et *action* peuvent être passés : par l'URL (cas d'un lien ou d'une redirection avec `header(Location:...)`) ou par un formulaire (au niveau de l'attribut `action` de la balise `<form>`).

☞ Testez les options du bandeau et observez l'URL:



Accueil	Nos produits par catégorie	Nos produits	Voir son panier
---------	----------------------------	--------------	-----------------

☞ Vous constatez que suivant le lien sélectionné, des paramètres *uc* et *action* sont envoyés au contrôleur principal `index.php` qui active un contrôleur spécifique en fonction de la valeur de *uc* qu'il traite grâce à un `switch`. Ensuite le contrôleur spécifique, traite l'action grâce au paramètre *action*.

Exemple quand l'utilisateur clique sur le lien « Nos produits par catégorie », l'URL du lien est `index.php ?uc=voirProduits&action=voirCategories`.

Dans le code du contrôleur principal (index.php), on a vu que si uc=voirProduits: c'est c_voirProduits.php qui est inclus et qui traitera le paramètre *action* reçu grâce à un switch qui teste l'action.

Dans le case 'voirProduits' du contrôleur c_voirProduits.php, la méthode getLesCategories() est appelée par l'instruction :

```
$lesCategories=getLesCategories()
```

puis grâce à un include du code de la vue v_categories.php, le contrôleur délègue à la vue le rôle de l'affichage du résultat stocké dans \$lesCategories.

Il récupère ensuite le paramètre *categorie* et appelle ensuite la méthode getLesProduitsDeCategorie() et délègue à la vue v_produitsDeCategorie.php l'affichage des produits stockés dans \$lesProduits.

```
case 'voirProduits' :  
{  
    $lesCategories = getLesCategories();  
    include("vues/v_categories.php");  
    $categorie = $_REQUEST['categorie'];  
    $lesProduits = getLesProduitsDeCategorie($categorie);  
    include("vues/v_produitsDeCategorie.php");  
    break;  
}
```

3.d Contrôleur et cas d'utilisation 'consulter les produits'

Ce cas propose deux sous-vues externes :

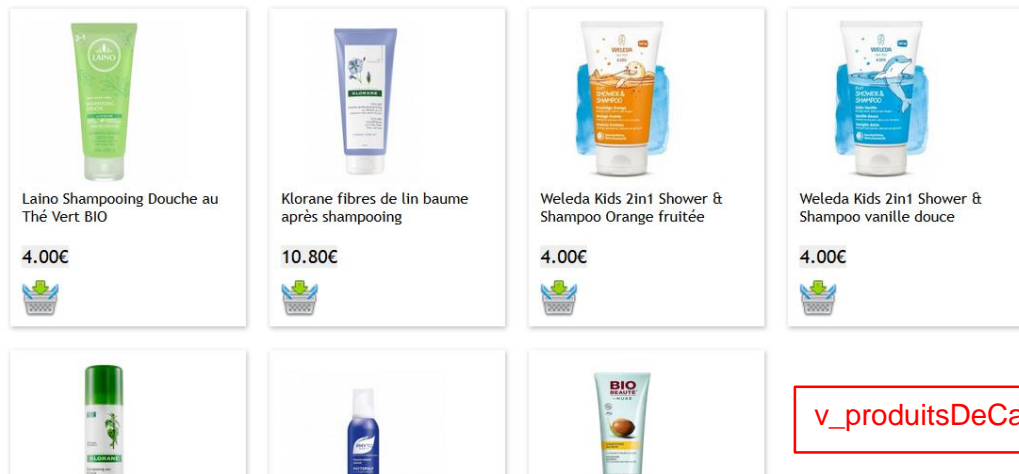
- l'affichage des catégories après avoir cliqué sur le lien 'Les produits par catégorie' du bandeau
- l'affichage des produits après avoir sélectionné une catégorie

La copie d'écran ci-dessous vous montre ces deux vues :

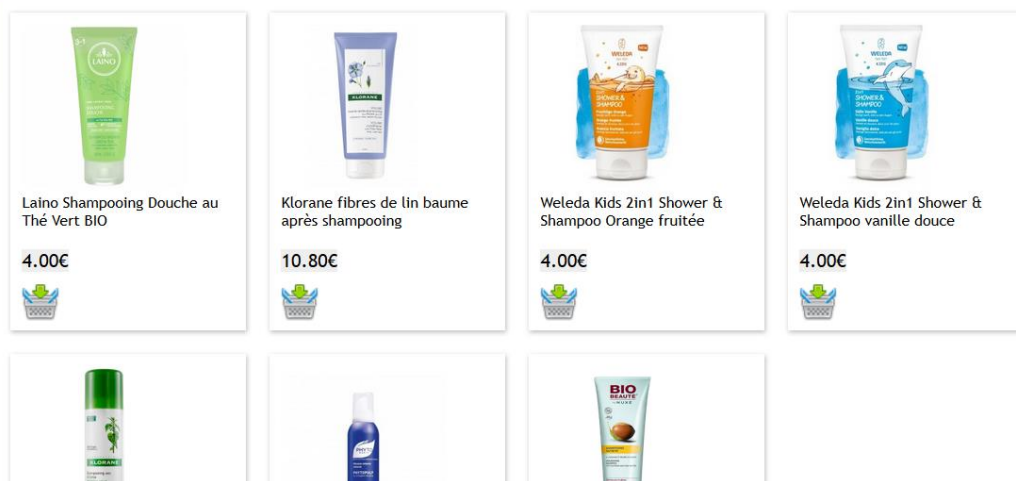
- A gauche : l'affichage des catégories grâce à v_categories.php
- A droite : l'affichage des produits après sélection de la catégorie grâce à v_produitsDeCategorie.php

Cheveux
Forme
Protection
Solaire

v_categories.php



Cheveux
Forme
Protection
Solaire



En fait, ce cas d'utilisation traite 3 cas :

- les deux premiers ci-dessus
- et un troisième correspondant au rechargement de la page après l'ajout d'un produit au panier.

C'est ainsi que le contrôleur c_voirProduits.php traite 3 actions : 'voirCategorie', 'voirProduits' et 'ajouterAuPanier'.

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"> Nos produits par catégorie </a></li>
```

Figure 5 - extrait du code du bandeau (cf v_bandeau.php)

Le code ci-dessus est extrait du bandeau (**v_bandeau.php**) : son rôle est prépondérant. Les liens qu'il contient permettent de passer à index.php par l'url le paramètre uc (sous la forme de `$_REQUEST[uc]`) et le paramètre action (sous la forme de `$_REQUEST[action]`).

Le paramètre uc fournit le cas d'utilisation, le paramètre action indique le traitement à effectuer dans le case.

Il en est de même pour le choix de la visualisation des produits après sélection de la catégorie avec un paramètre supplémentaire `categorie` qui permettra une restriction pour afficher uniquement les produits de la catégorie choisie.

```
<?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. Gestion des erreurs

Les erreurs liées à des saisies incorrectes sont signalées dans une vue distincte (`v_erreurs.php`) :

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

Figure 6 - code du fichier `v_erreurs.php`

Ce fichier est inclus si une erreur est décelée (il faut au préalable affecter le tableau `$msgErreurs` avec les messages d'erreur à afficher) :

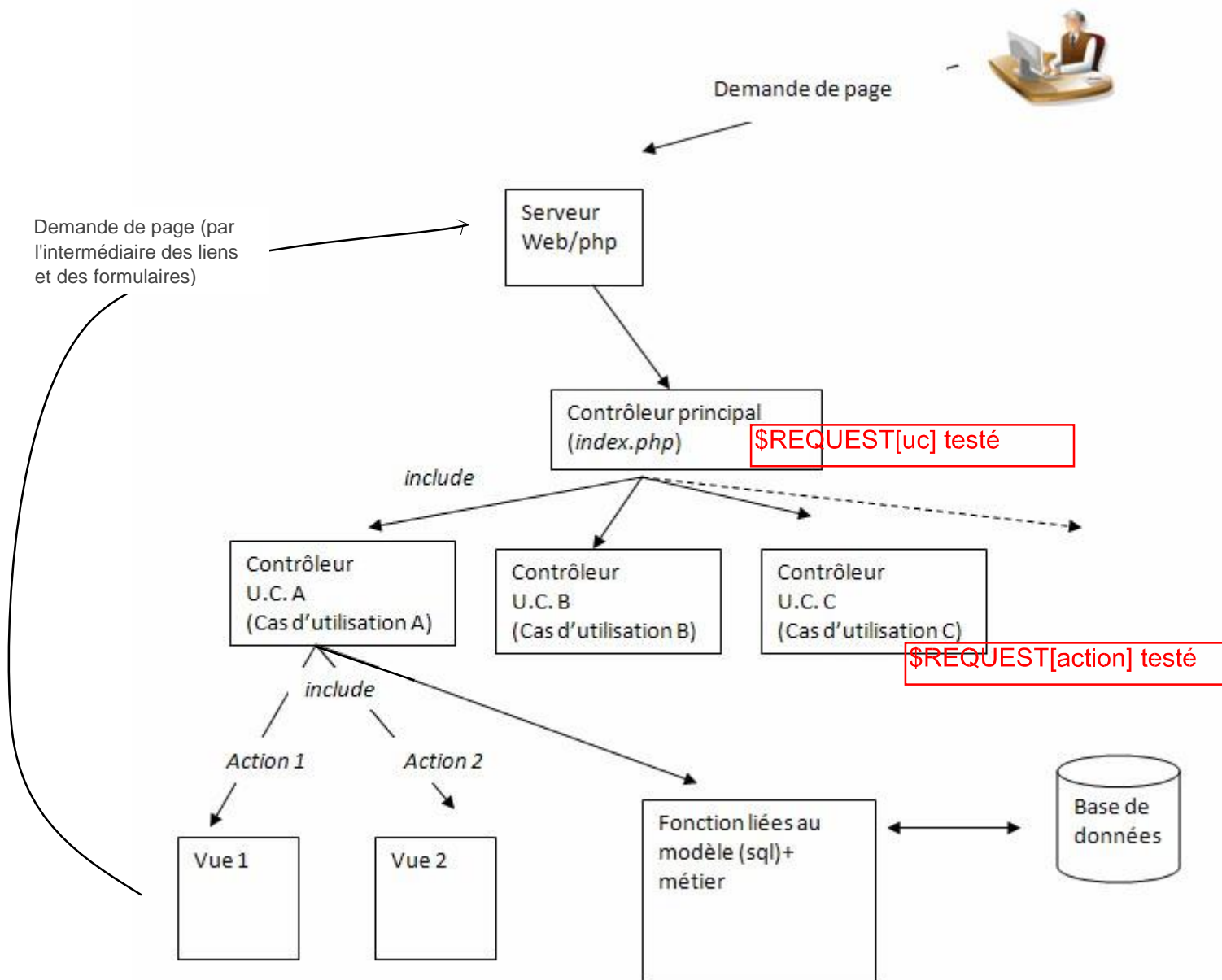
```
case 'confirmerCommande' :
{
    $nom=$_REQUEST['nom'];$rue=$_REQUEST['rue'];$ville=$_REQUEST['ville'];$cp=
    $msgErreurs = getErreursSaisieCommande($nom,$rue,$ville,$cp,$mail);
    if (count($msgErreurs) !=0)
    {
        include ("vues/v_erreurs.php");
        include ("vues/v_commande.php");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

La fonction `getErreursSaisieCommande` retourne le 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;
}
```

5. Synthèse

Voici un schéma résumant le fonctionnement d'une application en architecture MVC:



Seule la page index.php est chargée. Le reste des traitements est assuré grâce à des includes et à des appels aux méthodes de `bd.produits.onc.php` (qui gère le lien avec la base de données) ainsi qu'à des fonctions par exemple de gestion du panier (fichier `fonctions.inc.php`).

V. Git, un logiciel de gestion de versions

☞ Suivre le chapitre d'openClassroom <https://openclassrooms.com/fr/courses/7162856-gerez-du-code-avec-git-et-github/7165703-decouvrez-la-magie-du-controle-de-versions> pour en savoir plus sur ce qu'est un logiciel de gestion de versions (ou VCS en anglais, pour version control system).

VI. 1. Installer et configurer Git

VII. 1.a. Installation

Au lycée Git est installé sur C:\programmes.

Le cours openclassrooms contient un chapitre sur l'installation si vous voulez installer Git chez vous.

☞ Ouvrir la console de git bash (clic-droit sur le dossier GsbParam puis  Git Bash Here) qui vous permet de saisir les commandes git.

VIII. 1.b. configuration

☞ Tapez `git config --list` pour visualiser toutes les propriétés de votre environnement.

☞ Dans la console, chercher ces trois lignes :

```
color.diff auto
color.status auto
color.branch auto
```

☞ Si vous ne les trouvez pas, lancer les commandes :

```
git config --global color.diff auto
git config --global color.status auto
git config --global color.branch auto
```

Vous pouvez aussi ajouter la commande :

```
git config --global core.autocrlf false
```

qui évite de transformer des fins de ligne LF en CRLF.

Elles activeront la couleur dans Git. Il ne faut le faire qu'une fois, et ça aide à la lisibilité des messages dans la console.

--global : permet de définir les propriétés globalement pour tous vos répertoires suivis par GIT. Si vous l'omettez, les propriétés seront spécifiques au projet courant.

2. Créer un nouveau dépôt local

Nous allons créer dans notre dossier de travail un dépôt local qui stockera les versions successives du site.

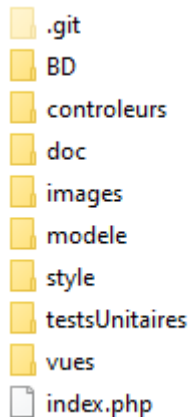
☞ dans la console Git avec le prompt qui indique que vous êtes dans le dossier GsbParam, tapez la commande `git init`

```
Marielle@DESKTOP-99H3C4R MINGW64 /c/wamp64/www/GsbParam
$ git init
```

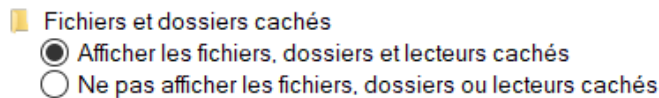
Qui vous retourne le message :

```
Initialized empty Git repository in C:/wamp64/www/testsGit/.git/
```

Ce message indique qu'un dossier .git a été créé dans votre dossier de travail (workspace git) qui contiendra toutes les versions du site.



Si le dossier .git n'est pas visible : actualisez l'explorateur ou vérifiez les options d'affichage (menu Affichage, options, modifier les options des dossiers et de recherche. Puis dans l'onglet Affichage, dans Paramètres avancés, vérifiez que 'afficher les fichiers... cachés' est coché).



Remarque : le . devant un nom de dossier ou de fichier indique un fichier ou dossier caché.

Le prompt de la console Git Bash est complété par `(master)` qui indique que votre workspace (dossier actuel) est versionné et qu'il s'agit de la branche de développement principale.

3. Configuration locale (pour ce projet)

☞ Il faut configurer votre nom (ou pseudo) :

```
git config --local user.name "votre_pseudo"
```

☞ Puis votre e-mail (indiquez un email valide, il nous servira ensuite) :

```
git config --local user.email "votre_email"
```

--local indique que la configuration est effectuée au niveau du repository donc de ce projet (comme vous êtes plusieurs à partager les postes c'est mieux).

4. Un premier commit du projet

IX. 4.a ajout des fichiers à la zone d'index

☞ toujours dans la console git Bash, tapez la commande `git add .` (le point est important, il désigne tous les fichiers comme * pour tous les champs en SQL).

Cette commande ajoute tous les fichiers du workspace (désignés par le .) à la zone Staging Area ou Zone d'index (qui contient les fichiers à inclure dans le versionning).

Les fichiers dans la staging area sont dit *staged* ou *indexés*.

Pour ajouter les fichiers un par un, on pourra faire `git add nomfichier`

X. 4.b Git status : observez le repository

☞ Utiliser **git status** : Cette commande vous indique où vous en êtes et donne des indications contextualisées sur ce que vous pouvez (ou devez) faire

```
Marielle@DESKTOP-99H3C4R MINGW64 /c/wamp64/www/GsbParam (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   BD/gsbpara_scriptBD.sql
    new file:   TestsUnitairesCorr/test_getLesCategories.php
    new file:   TestsUnitairesCorr/test_getLesProduitsDeCategori
    new file:   controleurs/c_gestionPanier.php
    new file:   controleurs/c_gestionProduits.php
    new file:   controleurs/c_voirProduits.php
    new file:   images/Thumbs.db
```

Pas encore de commit

Ce titre indique tous les changements à commiter

En vert, la liste de tous les fichiers nouveaux dans ...

XI. 4.c. Un premier commit

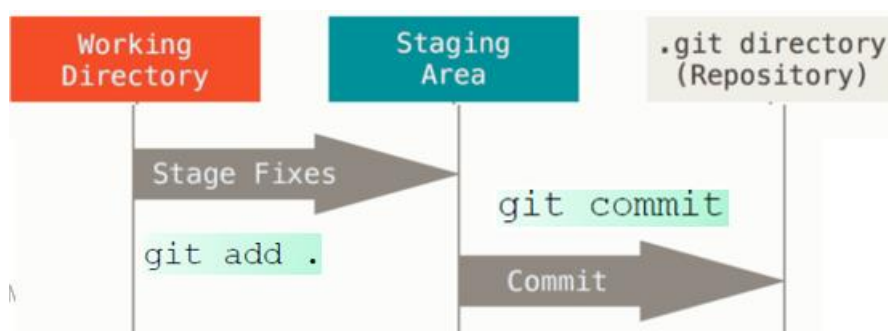
☞ toujours dans la console git Bash, tapez la commande

```
Marielle@DESKTOP-99H3C4R MINGW64 /c/wamp64/www/GsbParam (master)
$ git commit -m 'commit initial du projet'
```

qui permet de

passer les fichiers de la zone staging Area vers la zone versionnée du repository (dépôt local matérialisé par le dossier .git).

```
Marielle@DESKTOP-99H3C4R MINGW64 /c/wamp64/www/GsbParam (master)
$ git commit -m 'commit initial du projet'
[master (root-commit) d312266] commit initial du projet
54 files changed, 1086 insertions(+)
create mode 100644 BD/gsbpara_scriptBD.sql
create mode 100644 TestsUnitairesCorr/test_getLesCategories.php
create mode 100644 TestsUnitairesCorr/test_getLesProduitsDeCategori
create mode 100644 controleurs/c_gestionPanier.php
create mode 100644 controleurs/c_gestionProduits.php
create mode 100644 controleurs/c_voirProduits.php
```



Qu'est ce qui est sauvegardé ?

Chaque commit fait un instantané du projet. Il a un nom unique sous forme de hashcode ex : 64334a095b75811ab419e7d4f75bec2606d759dc . Ce nom est unique dans tous les dépôts : local ou distant.

Le commit ne contient que la différence par rapport au commit précédent (ce qui prend peu de place).

XII. 4.d. Fichiers unstaged et untracked

Un fichier *staged* s'il est modifié passe dans un état *unstaged*.

Un fichier nouveau et pas encore dans la *staging area* est dit *untracked*.

Pour ajouter les fichiers *unstaged* et *untracked* à la staging area, **il faut réaliser un git add**.

☞ Ajouter un commentaire dans index.php pour commenter le switch : `// traitement de l'uc : on charge le contrôleur approprié.`

☞ Après avoir sauvegardé le fichier, utiliser la commande **git status**.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.php

no changes added to commit (use "git add" and/or "git commit -a")
```

Vous pouvez voir le fichier qui a été modifié et on vous indique d'utiliser git add pour mettre à jour ce qui doit être versionné (commité), ou de faire un git restore pour annuler la modification et retrouver le fichier dans son état initial.

☞ Utiliser la commande **git restore index.php**. Puis la commande **git status**.

Vous remarquez que git vous indique qu'il n'y a rien à versionner.

☞ Ouvrez le fichier dans index.php pour constater qu'il est revenu dans son état initial.

☞ Ajouter de nouveau le commentaire pour le switch. Sauvegardez et cette fois réalisez le git add.

☞ Utiliser la commande **git status** (vous la retrouver en utilisant la flèche du haut dans la console). On vous indique qu'il faudra faire un commit de ce fichier modifié (ou un git restore pour le sortir de la staging area).

```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   index.php
```

Ne commitez pas tout de suite, le changement n'est pas assez important pour créer une nouvelle version.

4.e. Visualiser l'historique

Vous pouvez voir l'historique des commits et leurs caractéristiques (hashcode, auteur, date, heure, message) grâce à la commande `git log`.

☞ Afficher l'historique de votre workspace avec git log.

Vous pouvez aussi utiliser `$ git log --oneline` qui affiche un historique plus concis où les identifiants des commits sont réduits aux 7 premiers caractères.

Comme c'est un affichage assez pratique, faites-en un alias (vous pouvez faire un copier de cette commande et un clic droit dans la console puis paste) :

```
git config --global alias.lg « log --oneline »
```

Vous pouvez maintenant utiliser `git lg` pour à la place de `git log --online`.

4.f Evolution 1 : ajout fonctionnalité Nos produits par catégorie

☞ Modifier le code de l'application pour que lorsque l'utilisateur clique sur 'nos produits par catégorie', les produits de la catégorie CH (cheveux) soient affichés avec un titre 'Produits de la catégorie cheveux'. Si l'utilisateur choisit une catégorie à gauche, le titre changera et les produits de la catégorie choisie s'afficheront.

Vous avez remarqué que le lien du bandeau 'Nos produits', ne donne aucun résultat. On voit bien un paramètre `action=nosProduits` et une `uc=voirProduits` dans l'URL mais rien ne se passe.

☞ D'après l'url, qui est chargé de traiter l'action=`nosProduits` ? pourquoi ?

`voirProduits` de la page contrôleur `c_voirProduits.php`

☞ Compléter le code pour que l'action `nosProduits` soit traitée (elle affichera tous les produits). Indiquer dans votre compte-rendu votre démarche, ainsi que les modifications effectuées en précisant bien quels fichiers sont modifiés et le code ajouté.

```
function getLesProduits(){
    try
    {
        $monPdo = connexionPDO();
        $req = 'select `id`,`description`,`prix`,`image`,`idCategorie` from
`produit` ';
        $res = $monPdo->query($req);
        $lesProduits = $res->fetchAll(PDO::FETCH_ASSOC);
        return $lesProduits;
    }
    catch (PDOException $e)
    {
        print "Erreur !: " . $e->getMessage();
        die();
    }
}

case 'nosProduits': {
    $lesProduits = getLesProduits();
    include("vues/v_produits.php");
    break;
}
```

☞ Tester votre application en allant jusqu'à mettre un produit de la nouvelle page affichée dans le panier.

Cela fonctionne

☞ Quand tout est fonctionnel, utiliser `git status` pour avoir des informations sur votre dépôt.

☞ Réaliser un `git commit` avec un message indiquant 'ajout et test de Nos produits par catégorie'.

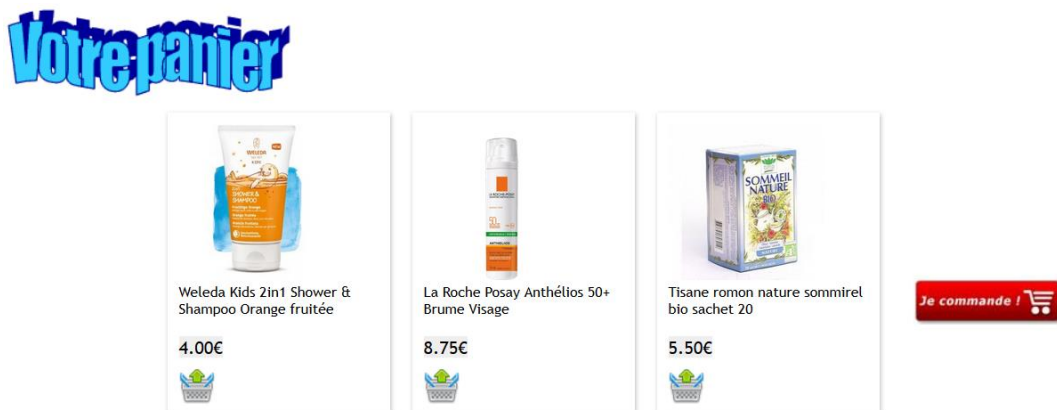
☞ Vérifier à l'aide d'un `git status` et `git lg` que tout est OK.

4 .g Evolution n°2 : cas d'utilisation 'gérer son panier'

Les vues externes

Les *vues externes* associées sont :

- Visualisation du panier



En cliquant sur le bouton Je commande!, on obtient un formulaire pour la saisie des informations du client nécessaires pour enregistrer la commande :

Commande

Nom Prénom*

rue*

code postal*

ville*

mail*

Valider

Annuler

Une fois validé, le formulaire peut être réaffiché avec les erreurs éventuelles :

• erreur de mail

Commande

Nom Prénom*

rue*

code postal*

ville*

mail*

Le contrôleur c_gestionPanier.php

☞ Observez le code du contrôleur.

☞ Combien de cas gère le contrôleur ? Indiquer quelle fonctionnalité se cache derrière chaque action (les fonctionnalités attendues sont expliquées dans la description du cas d'utilisation en annexe 1).

Il fait passer ces valeurs :

```
$nom
$rue
$ville
$cp
$mail
```

☞ Où se trouvent les fonctions qui gèrent le panier ?

Dans fonctions.inc.php

☞ Où est stocké le panier ? Coller ici le résultat d'un var_dump() qui montre la structure qui stocke les données du panier.

Dans le contrôleur c_gestionPanier.php

```
Var_dump['$_REQUEST'];
var_dump($lesProduitsDuPanier);
```

Tests de la fonctionnalité 'commander'

Travail à faire :

☞ Tester le bon fonctionnement du contrôleur en créant un panier et une commande. Vérifier le contenu de la BD. Apporter les corrections nécessaires aux erreurs trouvées.

Certains champs sont trop court et les paramètres passés peuvent être des injections.

```
$nom = '';
$nom = htmlspecialchars($nom);
$rue = '';
$rue = htmlspecialchars($rue);
$ville = '';
$ville = htmlspecialchars($ville);
$cp = '';
$cp = htmlspecialchars($cp);
```

```
$mail = '';  
$mail = htmlspecialchars($mail);
```

☞ Faites en sorte que le message 'commande enregistrée' ne soit pas systématiquement afficher (par exemple si l'insertion des informations en BD n'a pas fonctionné).

Amélioration :

- Ajouter une fonctionnalité 'vider le panier' (modifier la description du cas d'utilisation).

☞ Quand tout est fonctionnel, utiliser `git status` pour avoir des informations sur votre dépôt.

☞ réaliser un `git commit` avec un message indiquant 'ajout et test de Nos produits par catégorie'.

☞ Vérifier à l'aide d'un `git status` et `git lg` que tout est OK.

Pour avoir plus de détails sur un commit particulier, vous pouvez utiliser `git show` suivi de l'identifiant du commit (les 7 premiers caractères). Si vous ne précisez pas l'identifiant, le dernier commit sera concerné.

☞ Utiliser `git show` pour avoir plus de détails sur votre dernier commit.

Vous voyez apparaître dans l'éditeur, tous les fichiers modifiés (en vert les lignes ajoutées, en rouge les suppressions).

XIII. Git Travailler en remote (dépôt distant)

IMPORTANT :

Vous allez avoir la possibilité de récupérer chez-vous du code hébergé sur un serveur Git. Pour que vous disposiez d'un code à jour, et pour vous éviter des conflits de code, il est IMPORTANT de veiller à ce que :

- le code au lycée soit remonté vers le dépôt distant à la fin de chaque séance de TP
- Le code mis à jour chez-vous soit remonté vers le dépôt distant avant de vouloir le récupérer au lycée via le serveur Git

1) Qu'est-ce qu'un dépôt distant

L'intérêt de travailler en remote (à distance), c'est de pouvoir travailler à plusieurs sur un même dépôt distant ou de pouvoir travailler à domicile et au travail avec un même dépôt. Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau de l'entreprise donc ailleurs qu'en local (ailleurs que sur votre PC).

Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants. Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture. Collaborer avec d'autres personnes consiste à gérer ces dépôts distants, en poussant ou tirant des données depuis et vers ces dépôts quand vous souhaitez partager votre travail.

2) Où stocker un dépôt distant ?

Soit en réseau, soit sur Internet.

Il existe plusieurs hébergeurs de dépôts git :

- gitHub (très utilisé, il permet d'héberger des dépôts gratuits publics donc accessibles de tous ou privés). On trouve d'ailleurs sur gitHub un tas de projets connus comme par exemple [Symfony](#), la nouvelle version du framework PHP qui permet de créer des sites robustes facilement (il s'agit ici de la version *Symfony* en cours de développement).
- bitBucket qui permet de gérer des dépôts privés gratuitement.

Nous commencerons par gitHub.

3) Un premier dépôt distant sur gitHub

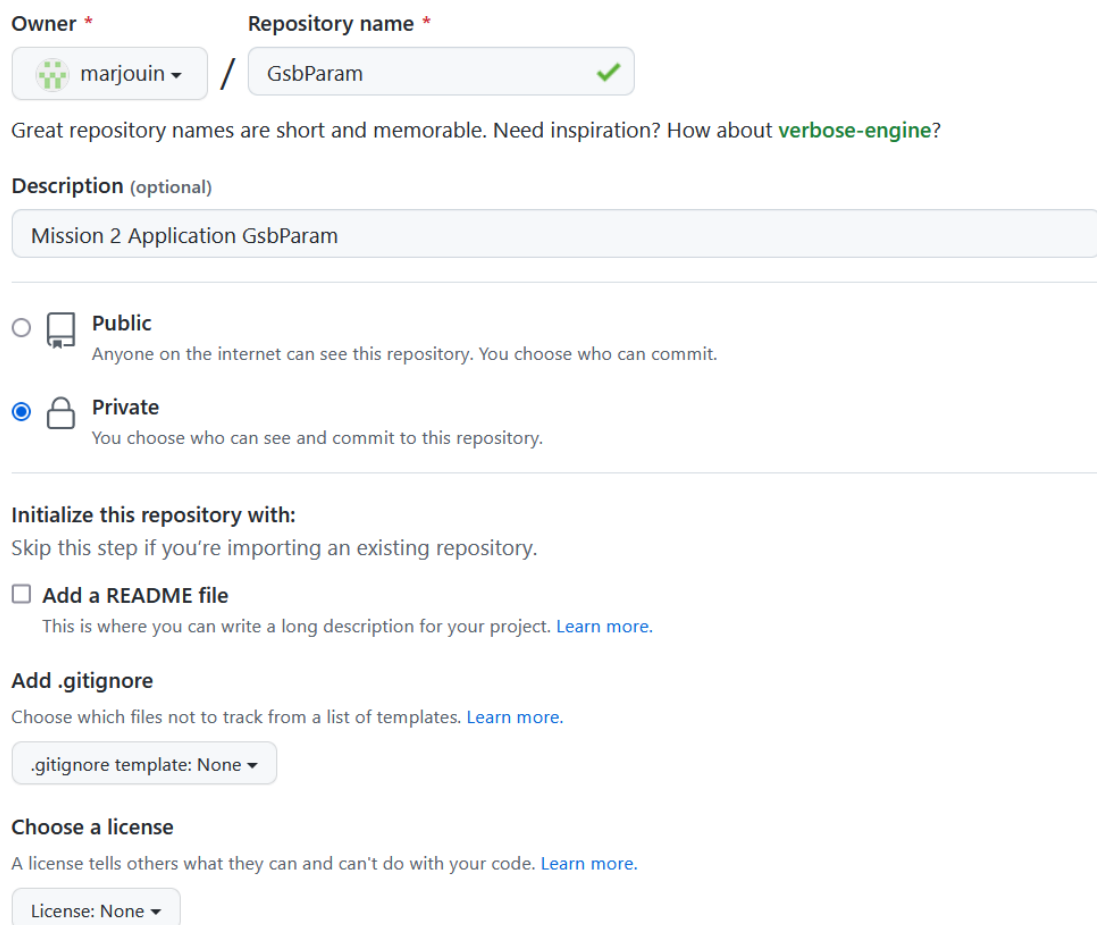
☞ Il faut d'abord créer un compte sur le site de github : c'est aussi simple que de s'enregistrer sur n'importe quel réseau social. Utilisez l'e-mail que vous avez indiqué lors du git config.

☞ Il faut d'abord **créer un repository** à l'aide du bouton

Repositories



Voici ce que j'ai indiqué (ne créer pas de README file).

A screenshot of the GitHub repository creation form. The form has two main sections: "Repository name" and "Description". The "Repository name" section has a dropdown menu for the owner (selected "marjoun") and a text input for the repository name (containing "GsbParam" with a green checkmark). Below this is a link for more inspiration: "Great repository names are short and memorable. Need inspiration? How about [verbose-engine?](#)". The "Description" section has a text input containing "Mission 2 Application GsbParam". Below the description is a section for "Initialize this repository with:" which includes a "Public" option (selected) and a "Private" option. The "Public" option has a description: "Anyone on the internet can see this repository. You choose who can commit." The "Private" option has a description: "You choose who can see and commit to this repository." Below this is a section for "Add a README file" with a checkbox and a link "Learn more.". Then there is a section for "Add .gitignore" with a dropdown menu for ".gitignore template:" (selected "None") and a link "Learn more.". Finally, there is a section for "Choose a license" with a dropdown menu for "License:" (selected "None") and a link "Learn more.".

☞ Valider en cliquant sur le bouton Create Repository

A la création du dépôt, vous pouvez voir l'URL de votre dépôt distant et les commandes pour pousser le code de votre dépôt local vers le dépôt distant :



Vous pouvez aussi avoir accès à ces URL en cliquant sur le bouton **<> Code** à chaque fois que vous allez dans les settings de votre repository.

4) Git et les protocoles réseaux (https et ssh)

Les URL d'un dépôt distant diffèrent suivant le protocole de communication entre le client et le serveur Git choisi : il y a deux protocoles ssh et https.

Voici la présentation, les avantages et les inconvénients de chaque protocole :

4.a. Le protocole SSH

Le protocole SSH permet de communiquer à l'aide de 2 clés (1 publique et 1 privée). Il s'agit d'un protocole permettant à un client (un utilisateur ou bien même une machine) d'ouvrir une session interactive sur une machine distante (serveur) afin d'envoyer des commandes ou des fichiers de manière sécurisée. Le client et le serveur s'authentifient mutuellement afin d'assurer que les deux machines sont bien celles qu'elles prétendent être. Ce protocole utilise le port 22 par défaut.

Avantages

L'accès distant à travers SSH est sécurisé, toutes les données sont chiffrées et authentifiées. Enfin, SSH permet de compresser autant que possible les données avant de les transférer.

Inconvénients

Le point négatif avec SSH est qu'il est impossible de proposer un accès anonyme au dépôt. Les accès sont régis par les permissions SSH, même pour un accès en lecture seule, ce qui s'oppose à une optique open source.

Mise en œuvre : Génération des clés (à faire)

- Ouvrir gitBash.
- Saisir la commande `ssh-keygen` (vous pouvez copier cette commande et grâce à un clic droit dans gitBash effectuer un paste) :

Résultat : on vous demande de confirmer le fichier de stockage des clés (dans cet exemple `z:/.ssh/id_rsa` mais vous pouvez accepter le stockage dans `/C:/users/`)

```
Generating public/private rsa key pair.
Enter file in which to save the key (/z//.ssh/id_rsa):
```

(.ssh est un dossier caché comme tous les dossiers dont le nom est précédé par un .)

Noter ici l'endroit où sera stockée votre paire de clés, si vous choisissez un autre lieu de stockage :

- c) Confirmer en validant. Ensuite on vous demande une passphrase qui vous sera demandée dans gitHub quand vous voudrez accéder au dépôt :

Enter passphrase (empty for no passphrase):

- d) Saisir et confirmer une phrase secrète (qui sera votre mot de passe). Résultat :

```
Your identification has been saved in /z//.ssh/id_rsa.
Your public key has been saved in /z//.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ILgfB3em+EqCYCNHVSu+LakOJJ3t9MVkpQAE/+uf+C4 jouin@J205-17
The key's randomart image is:
+---[RSA 2048]---+
|.oo ...
|. + .
|+ = + +
|o = B O
|+= B = S
|*o.+ @ o
|o . O =
|. =E+ .
|.o +==
+---[SHA256]---+
```

Chemin vers la

Le système affiche l'empreinte de votre paire de clés et son image.

- e) Transférer la clé **publique** contenue dans fichier de clés id_rsa.pub vers le buffer en entrant la commande :

clip < /z//.ssh/id_rsa.pub

Chemin à remplacer par celui
affiché à l'étape précédente (sans
la fin)

Résultat : vous allez pouvoir copier votre clé publique SSH du buffer vers gitHub

- f) Dans GitHub, dans le menu de votre profil (accessible en haut à droite grâce à une icône), dans *Settings*, choisir SSH and GPG KEYS, puis *new SSH key*, saisir un libellé et faire un coller (ctrl+V) dans la zone de clé juste au-dessous. Puis valider l'ajout de la clé.
- g) Pour cloner (récupérer le code d'un dépôt distant) ou faire un push (remonter du code local dans le dépôt distant), vous utiliserez ensuite l'URL SSH de votre dépôt distant. Ce que vous allez faire au paragraphe qui suit l'explication sur https.

4.b Le protocole HTTP(S)

introduction

Le HTTP(S) est un protocole de transfert des données entre un client et un serveur sur le web. Un serveur HTTP utilise par défaut le port 80 et un serveur HTTPS le port 443. On peut cloner un répertoire comme suit :

```
git clone https://git-server-address/username/repository
```


Il faut noter que lors d'une requête HTTP(S), on est souvent amené, là où c'est nécessaire (git clone, git fetch, git pull, ou git push), à saisir son identifiant et mot de passe pour terminer l'opération.

Inconvénient

Ne fonctionne pas pour le git push, si vous accédez à internet via un proxy ou dans certains EDI comme Eclipse.

Mise en œuvre.

- a) Sur le site de github, il faut vous créer un Personal Access Token : voir cet article <https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token>

Le PAT créé peut avoir une durée de 30 jours. On pourra en recréer par la suite.

- b) Pensez, en plus de cocher la rubrique repo, à cocher aussi la rubrique admin :repo_hook. Au moment de faire le git push, on vous demandera de vous authentifier, utiliser ce PAT à la place du mot de passe.

Conclusion

Les deux protocoles servent à communiquer avec le serveur Git. Toutefois, le HTTP(S) requiert de saisir les identifiants pour certaines requêtes, tandis que le SSH utilise des clés publique et privée. Tandis que le SSH utilise le port 22 par défaut, le HTTP(S) utilise les port 80 et 443, des ports qui sont rarement bloqués sur un réseau contrairement au port 22. Le choix de votre protocole doit donc dépendre de ces contraintes du réseau sur lequel vous êtes connectés.

Au cas où vous devez changer de protocole, il suffit de changer l'adresse du serveur distant dans vos paramètres Git (voir modèles d'URL en début de document).

5) Remonter le code vers le repository distant

Comme gitHub vous le propose, vous allez remonter le code du projet local vers le dépôt distant.

5.a. Ajouter un dépôt distant

Au niveau local, il faut faire le lien vers le dépôt distant grâce à la commande `git remote add [nomcourt] [url]` : nomcourt = nom court du dépôt distant, ici **origin** car c'est le seul dépôt vers lequel on peut 'pousser', cad envoyer son code. On peut ajouter d'autres dépôts distants avec d'autres noms mais vers lesquels on ne pourra pas pousser du code mais juste le récupérer (cloner).

☞ Réaliser le git remote add origin en indiquant l'URL SSH de votre dépôt distant. GitHub vous l'indique dans les propositions faites sous les URL de votre dépôt comme le montre la copie d'écran ci-dessus (utiliser SSH) :

```
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:marjouin/GsbParam.git
git push -u origin main
```

5.b. Pousser le code de `www` vers le dépôt distant : `git push`

Comme le protocole `https` n'est pas utilisable au lycée pour faire un `push`, vous allez devoir générer votre paire de clés.

☞ il faut avoir au préalable générer les clés SSH et ajouter la clé publique à vos settings (voir paragraphe mise en œuvre SSH ci-dessus).

Un `git push` doit être réalisé, lorsque votre dépôt vous semble prêt à être partagé (soit pour le rendre disponible pour l'équipe qui travaille sur le projet, soit pour vous permettre de continuer chez vous, par exemple). La commande pour le faire est simple : `git push [nom-distant] [nom-de-branche]`. Si vous souhaitez pousser votre branche `master` vers le serveur `origin`, alors vous pouvez lancer ceci pour pousser votre travail vers le serveur amont, la commande sera `git push origin master`.

☞ Réaliser un `push` avec la commande `git push origin master`

☞ Saisir la passphrase utilisée pour générer vos clés.

```
$ git push origin master
Enter passphrase for key '/c/Users/Marielle/.ssh/id_rsa':
Enumerating objects: 62, done.
Counting objects: 100% (62/62), done.
Delta compression using up to 4 threads
Compressing objects: 100% (59/59), done.
Writing objects: 100% (62/62), 506.98 KiB | 1.70 MiB/s, done.
Total 62 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To github.com:marjouin/GsbParam.git
 * [new branch]      master -> master
```

Remarque : Quand vous travaillerez en collaboration, cette commande `git push` pourrait ne pas fonctionner. Par exemple, si vous n'avez de droits d'accès en écriture sur le dépôt distant ou si un collaborateur a poussé son code avant vous. Dans ce dernier cas, vous devrez tout d'abord tirer les modifications de l'autre personne et les fusionner avec les vôtres avant de pouvoir pousser.

Comme c'est le premier `push` et que vous travaillez seul, pour le moment, seul un dépôt local non mis à jour après des modifications de votre code peut vous bloquer.

☞ Vérifier sur `gitHub` en actualisant la page, l'état de votre repository, vous pourrez y voir les fichiers conservés dans le dépôt distant (repository) et en cliquant sur le lien indiquant le nombre de commit voir les différents commits réalisés précédemment en local et leurs messages.

marjouin **GsbParam** private

<> Code Issues Pull requests Actions Projects Security Insights Settings

master 1 branch 0 tags Go to file Add file Code

marjouin commit initial du projet		d312266 1 hour ago 1 commit
BD	commit initial du projet	1 hour ago
TestsUnitairesCorr	commit initial du projet	
controleurs	commit initial du projet	
images	commit initial du projet	
modele	commit initial du projet	1 hour ago
vues	commit initial du projet	1 hour ago
index.php	commit initial du projet	1 hour ago

6) Lister les dépôts distants

`git remote -v` permet de visualiser les nom courts et les URL des dépôts distants.

7) Supprimer les références vers les dépôts distants

Si vous avez indiqué une mauvaise url pour origin, il faudra supprimer les références au dépôt distant erronées.

Vous utiliserez la commande : `git remote rm origin`

8) Récupération d'une copie locale du contenu du repository : git clone

Git clone est la commande utilisée pour récupérer une copie locale du dépôt distant, ce qui est le cas quand vous travaillez sur plusieurs PC (donc à faire chez vous, pour continuer le projet débuté au lycée, ou au lycée si vous changez de PC). Cette commande permet de recréer le dossier du projet et son contenu, le tout avec un historique de suivi.

☞ Démarche à faire chez vous : à partir de `www`, ouvrir la console Git bash et effectuer la commande `git clone url_du_repository` (`url_du_repository` à remplacer par l'url du dépôt). Chez vous, vous pouvez utiliser le protocole HTTPS.

☞ une fois le clone réalisé, fermer la console Git Bash du dossier `www`, pour ouvrir une console Git Bash sur le dossier `GsbParam` cloné.

Rappelez-vous :

Pour que vous disposiez d'un code à jour, et pour vous éviter des conflits de code, veillez à ce que :

- le code au lycée soit remonté vers le dépôt distant avant la fin de la séance de TP (voir paragraphe mettre à jour le dépôt distant à partir du dépôt local).
- Le code mis à jour chez-vous soit remonté vers le dépôt distant avant de vouloir le récupérer au lycée en séance de TP via le serveur Git (voir paragraphe, mettre à jour le dépôt local à partir du dépôt distant).

XIV. Autres évolutions demandées.

Toutes les évolutions demandées ci-dessous, devront être versionnées avec Git et remontées dans le dépôt distant. Le travail sera un travail individuel qui sera évalué (la gestion de version avec Git sera intégrée dans l'évaluation).

1ère partie :

1.1) Vérifier l'utilisateur utilisé pour la connexion à la BD : vous ne devez pas utiliser root. Si c'est le cas, créer un utilisateur avec un mot de passe et lui donner des droits limités (pas de droits globaux) pour qu'il puisse être utilisé pour la connexion à la BD à la place de root.
dev :dev

1.2) Les clients réguliers sont peu satisfaits du formulaire de commande : il faut leur faciliter la tâche en leur proposant de se connecter (pour cela ils doivent s'inscrire).

Réflexion à mener :

- Quel est l'impact sur la base de données ?
- Quel est l'impact sur le formulaire de commande ?

A faire :

- a) Créer un cas d'utilisation Se connecter (qui permettra aussi de se déconnecter)
- b) Modifier le cas d'utilisation 'gérer le panier' de l'annexe 1 pour tenir compte d'un utilisateur connecté ou pas
- c) Préparer un MCD montrant les évolutions nécessaires de la BD, à faire valider
- d) Modifier la base de données.
- e) Modifier le code de l'application. Pensez à sécuriser la connexion !
- f) Tester (indiquez dans un fichier tests à stocker dans le dossier tests unitaires, tous les tests réalisés pour vérifier que l'application fonctionne correctement). Structurer votre fichier.

2ème partie

Actuellement, l'internaute ne peut mettre dans son panier qu'un seul exemplaire d'un article. On souhaite pouvoir offrir la possibilité de mettre plusieurs exemplaires d'un article.

Pour cela, il faudra changer la base de données.

2.1) Modifier la base de données et son script de création.

2.2) On vous fournit le cas d'utilisation "gérer le panier" modifié par cette nouvelle règle de gestion (voir annexe 1). Modifiez l'application pour respecter ce cas d'utilisation modifié (vous pouvez utiliser du JavaScript).

2.3) Tester (compléter le fichier tests pour vérifier que la 2ème partie fonctionne correctement).

2.4) Réaliser des tests de non régression pour vérifier que les modifications apportées à la 2ème partie n'ont pas dégradé le fonctionnement de l'application constaté à la fin de la première partie).

3ème partie (à réaliser uniquement si 1ère et 2ème partie fonctionnent correctement)

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 en annexe 2 et en suivant la démarche proposée.

Remarques, pour des raisons de sécurité :

- une table **Administrateur** a été créée et devra contenir ses deux enregistrements :

```
INSERT INTO administrateur (id, nom, mdp) VALUES
('1', 'LeBoss', 'TheBest$147#'),('2', 'LeChefProjet', 'NearlyTheBest$280@');
```

- On ne souhaite pas l'ajout d'un bouton administrer au menu (la fonctionnalité administrer doit rester confidentielle).

3) Dessiner les vues externes et coder les pages nécessaires : vous pouvez bien sûr améliorer la sécurité en hashant ces mots de passe (sans les modifier).

Annexe 1 - Cas d'utilisation : 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 modifications.

Annexe 2 - Cas d'utilisation : gérer les produits

Acteur : l'administrateur

Scénario normal (le plus fréquent)

1. L'administrateur demande à se connecter
2. Le système demande le login et le mot de passe.
3. L'internaute saisit le nom login 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 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, 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. À 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.