Code sous forme Texte

I. Controllers :

1. DBController :

```php
<?php

/*
  Dans ce qui suit, nous construisons une classe pour gérer les connexions
et les requêtes faites à la bdd.
  C'est une classe ayant pour attribut un objet de la classe DBController,
et qui sera
  dotée de deux méthodes :
 - openConnection(): pour ouvrir une connexion avec la bdd
 - closeConnection() : pour fermer la connexion avec la bdd
 - select(), insert(), delete() pour exécuter des requêtes avec la bdd

 Le but est d'avoir un outil compact et fiable, nous permettant facilement
de dialoguer avec la bdd

 Le choix est fait d'utiliser l'extension mySQLi (mySQL improved) comme
interface d'accès à la bdd
  */


class DBController
{
    // identifiants et password
    public $dbHost="mysql02.univ-lyon2.fr";
    public $dbUser="php_memansour";
    public $dbPassword="HT_nNV6QW/8Xvky/L633pt2lp";
    public $dbName="php_memansour";

    // attribut d'instance
    public $connection;

    /*
     Methode openConnection(), ne prend pas de paramètre, retourne un
boolean.
     Permet d'ouvrir une connexion avec la bdd
     */
    public function openConnection()

    {
        // établir la connexion avec une nouvelle instance de mysqli, et
les paramètres de la bdd
        $this->connection=new mysqli($this->dbHost,$this->dbUser,$this-
>dbPassword,$this->dbName);

        // si un problème de connexion est survenu
        if($this->connection->connect_error)
        {
            // afficher un message dans la console et retourner false
            echo " Erreur de connexion : ".$this->connection-
```

```
>connect_error;
            return false;
        }
        else
        {
            // sinon la connexion est établie, la fonction s'arrête en
retournant true
            return true;
        }

    }

    /*
     Methode closeConnection(), ne prend pas de paramètre, ne retourne pas
de valeur.
     Permet de fermer une connexion avec la bdd
     */
    public function closeConnection()
    {
        if($this->connection)
        {
            // Si une connexion est en coursn elle est fermée par la
méthode close de la classe mysqli
            $this->connection->close();
        }
        else
        {
            // s'il n'y a pas de connexion, on affiche un message dans la
console
            echo "Pas de connexion à fermer !";
        }
    }


    /*
     Methode select(), prend en paramètre une chaine de caractères qui
represente une requête en langage SQL,
     retourne false ou les lignes récupérées de la bdd. elle est utilisée
quand une connexion est déjà établie avec la bdd
     */
    public function select($qry)
    {
        // On soumet la requête avec la methode query() de mysqli
        $result=$this->connection->query($qry);

        // Si pas de résultat
        if(!$result)
        {
            // on récupère l'erreur grâce à la méthode mysqli_error, et on
l'affiche dans la console, la fonction s'arrête en retournant false
            echo "Erreur : ".mysqli_error($this->connection);
            return false;
        }

        // S'il y a bien un résultat, on les récupère grâce à la méthode
fetch_all appliqué au résultat $resultat, la fonction s'arrête en
retournant les lignes récupérées
        else
        {
             return $result->fetch_all(MYSQLI_ASSOC);
        }
```

```
    }

    /*
     Methode insert(), prend en paramètre une chaine de caractères qui
représente une requête d'insertion en langage SQL,
     retourne false ou l'id de la dernière insertion. Elle est utilisée
quand une connexion est déjà établie avec la bdd
     */
    public function insert($qry)
    {
        $result=$this->connection->query($qry);
        if(!$result)
        {
            echo "Erreur : ".mysqli_error($this->connection);
            return false;
        }
        else
        {
             return $this->connection->insert_id;
        }

    }

    /*
     Methode delete(), prend en paramètre une chaine de caractères qui
représente une requête de suppression en langage SQL,
     retourne un boolean. Elle est utilisée quand une connexion est déjà
établie avec la bdd.
     */
    public function delete($qry)
    {
        $result=$this->connection->query($qry);
        if(!$result)
        {
            echo "Erreur : ".mysqli_error($this->connection);
            return false;
        }
        else
        {
             return $result;
        }

    }

}
```

2. AuthController :

```
<?php
/*
  Dans ce qui suit, nous construisons une classe pour controler les
processus d'authentification.
  C'est une classe ayant pour attribut un objet de la classe DBController,
et qui sera
  dotée de deux méthodes :
 -login()
 -register()

 Le but est d'avoir un outil compact et fiable, nous permettant facilement
```

```
de réaliser les opérations de logging et
 d'enregisterement
  */



// Récupérer les outils déjà construits
require_once '../../Models/user.php' ;
require_once '../../Controllers/DBController.php' ;


class AuthController
{

    protected $db;

    /*
     Methode login(), prend un objet User en paramètre, retourne un
boolean.
     Permet de vérifier l'existance d'un utilisateur dans la base de
données
     */
    public function login(User $user)
    {
        // on affecte à l'attribut $db d'un objet de cette classe, la
valeur "un objet de la classe DBController" fraichement instancié
        $this->db=new DBController;

        // on fait appel à la méthode openConnection() de $db pour établir
une connexion avec la bdd
        if($this->db->openConnection())
        {
            /*
            Si la connexion est établie, on construit une requête SQL avec
les valeurs des attributs email et password
            du paramètre, requête pour récupérer dans la base de données
tous les attributs de l'utilisateur qui aura l'email
            et le password de l'objet $user

            */
            $query="select * from users where email='$user->email' and
password ='$user->password'";

            // On execute cette requête en faisant appel à la méthode
select() de l'objet $db, et on récupère le résultat
            $result=$this->db->select($query);

            // Plusieurs cas de figures possibles:

            // Si l'execution de la requête retourne false
            if($result===false)
            {
                // On affiche un message d'erreur dans la console
                echo "Erreur de requête !";

                // et la fonction s'arrête en retournant false
                return false;
            }

            // si le résultat est différent de false, la requête s'est bel
et bien exécutée, étudions la réponse de la bdd
```

```
            else
            {
                // Aucune ligne dans la table users de la bdd, ne contient
l'usager spécifié en paramètre
                if(count($result)==0)
                {
                    session_start();

                    // on stoque dans la variable $_SESSION un message
d'erreur en conséquence
                    $_SESSION["errMsg"]="Identifiant ou mot de passe
erronnés !";

                    // on ferme la connexion
                    $this->db->closeConnection();

                    // et la fonction s'arrête en retournant false
                    return false;
                }

                // S'il y a bien un usager repertoriée dans la bdd avec le
mail et le password du paramètre $user
                else
                {
                    session_start();

                    // ayant récupéré tous les attributs de cet usager,
j'affecte son id et son nom à la session
                    $_SESSION["userId"]=$result[0]["id"];
                    $_SESSION["userName"]=$result[0]["name"];


                    // Pour le rôle, qui a été récupéré sous forme d'un
chiffre, je l'affecte à la session sous forme de "Client" ou "Admin"
                    // cela permet de faciliter l'affichage du statut de
l'usager dans la vue (ne pas avoir à chaque fois à convertir)
                    if($result[0]["roleid"] == 2 )
                    {
                        $_SESSION["userRole"]="Client";
                    }
                    else
                    {
                        $_SESSION["userRole"]="Admin";
                    }

                    // Systématiquement fermer la connexion à la fin du
besoin
                    $this->db->closeConnection();

                    // et la fonction login() s'arrête en retournant true
                    return true;
                }
            }
        }

        // Si openConnection() rencontre un problème lors de son execution
        else
        {
            // on affiche un message d'erreur dans la console
            echo "Erreur de connexion à la base de données !";
```

```
            // et la fonction s'arrête en retournant false
            return false;
        }
    }

    /*
     Methode register(), prend un objet User en paramètre, retourne un
boolean.
     Permet d'enregistrer un nouvel utilisateur dans la base de données
     */

    // Les commentaires seront plus brefs vu les similarités avec login() !
    public function register(User $user)
    {
        $this->db=new DBController;

        // Si la connexion avec la bdd est établie
        if($this->db->openConnection())
        {
            // on construit une requête d'insertion avec les attributs de
l'objet $user
            $query="insert into users values ('','$user->name','$user-
>email','$user->password',2)";

            // on exécute cette requête grâce à la méthode insert(), et on
récupère le résultat
            $result=$this->db->insert($query);

            /*
             Si l'insertion est bel et bien executée, l'usager est
désormais repertorié dans la bdd, dispose d'un compte client
            et passe en mode connecté, une session est ouverte aux
caractéristiques de l'usager.
            Notons que l'enregistrement est par defaut uniquement pour les
clients, nous pouvons imaginer
            que les administrateurs soient créés directement dans la bdd
par un super Admin.
            La connexion est ensuite fermée et la fonction s'arrête en
retournant true.
             */
            if($result!=false)
            {
                session_start();
                $_SESSION["userId"]=$result;
                $_SESSION["userName"]=$user->name;
                $_SESSION["userRole"]="Client";
                $this->db->closeConnection();
                return true;
            }
            /*
             Si l'insertion n'a pas pu s'exécuter correctement, on fait
remonter l'erreur dans la variable $_SESSION,
             on ferme la connexion et la fonction s'arrête en retournant
false
            */
            else
            {

                $_SESSION["errMsg"]="Une erreur est survenue.. réessayer
plus tard !";
                $this->db->closeConnection();
```

```
                return false;
            }
        }

        /*
         Si la connexion n'a pas pu s'établir, on affiche un message dans
la console et la fonction s'arrête en retournant false.
         */
        else
        {
            echo "Erreur de connexion à la base de données !";
            return false;
        }
    }

}
```

3. ProductController :

```
<?php
/*
  Dans ce qui suit, nous construisons une classe pour controler les
processus d'authentification.
  C'est une classe ayant pour attribut un objet de la classe DBController,
et qui sera
  dotée de 6 méthodes qui interrogent la base de données et récupèrent la
réponse.

 - getCategories() : pour avoir la liste des catégories existantes dans la
bdd
 - getAllProducts() : pour avoir la liste des produits existants dans la
base de données, sans l'attribut image
 - getAllProductsWithImages() : pour avoir la liste des produits existants
dans la base de données, avec le chemin d'accès à l'image
 - getCategoryProducts() : pour avoir tous les produits d'une categorie
dont l'id est rentré en paramètre
 - addProduct() : pour ajouter une ligne produit dans le tableau produits
de la base de données
 - deleteProduct() : Pour supprimer une ligne produit dans le tableau
produits de la bdd


 Le but est d'avoir un outil compact et fiable, nous permettant facilement
de réaliser des opérations de lecture et d'insertion
 dans la base de données.
  */

// Commençons par récupérer nous outils
require_once '../../Models/product.php' ;
require_once '../../Controllers/DBController.php';

class ProductController
{
    protected $db;

    /*
    Methode getCategories(), ne prend pas de paramètre, retourne false ou
l'array des lignes de catégories obtenues
    suite à l'interrogation de la bdd
```

```
     */
    public function getCategories()
    {
        // instancier un nouvel objet DBController et l'affecter à
l'attribut $db
         $this->db=new DBController;

         // faire appel à openConnection()
         if($this->db->openConnection())
         {
             // Si la connexion est établie

             // Selectionner toutes les lignes du tableau categories de la
bdd et les retourner en résultat
            $query="select * from categories";
            return $this->db->select($query);
         }

         // Si un problème de connexion survient
         else
         {
             // le signaler dans la console et retourner false
            echo "Error in Database Connection";
            return false;
         }
    }

    /*
     Notons pour la suite, pour éviter les redondances de commentaires,
    toutes ces petites méthodes fonctionnent sur le même principe:
    - établir une connexion avec la bdd
    - exécuter une requête sql
    - retourner le résultat, et le message approprié

    L'utilisation des méthodes de la classe DBController, est plus
détaillée dans les commentaires de la classe AuthController
     */

    /*
    Methode addProduct(), permet d'insérer une nouvelle ligne dans le
tableau produit de la bdd.
    Prend en paramètre un objet Produit, retourne false ou l'id du dernier
produit inséré.
     */
    public function addProduct(Product $product)
    {
         $this->db=new DBController;
         if($this->db->openConnection())
         {
            $query="insert into products values ('','$product-
>name','$product->description','$product->price','$product-
>quantity','$product->image',$product->categoryid)";
            return $this->db->insert($query);
         }
         else
         {
            echo "Error in Database Connection";
            return false;
         }
    }
```

```
    /*
    Methode getAllProducts(), ne prend pas de paramètre, retourne false ou
l'array des lignes de la table produits de la base de données, sans
l'attribut image.
    suite à l'interrogation de la bdd
     */
    public function getAllProducts()
    {
         $this->db=new DBController;
         if($this->db->openConnection())
         {
            $query = "SELECT products.id, products.name, products.price,
products.quantity, categories.name AS `category` FROM products, categories
WHERE products.categoryid = categories.id";

            return $this->db->select($query);
         }
         else
         {
            echo "Error in Database Connection";
            return false;
         }
    }

    /*
    Methode deleteProduct(), permet de supprimer une ligne de la table
produits de la bdd.
    Prend en paramètre l'id du produit à supprimer, retourne un boolean.
     */
    public function deleteProduct( $id)
    {
         $this->db=new DBController;
         if($this->db->openConnection())
         {
            $query="delete from products where id = $id";
            return $this->db->delete($query);
         }
         else
         {
            echo "Error in Database Connection";
            return false;
         }
    }
        /*
        Methode getAllProductsWithImages(), ne prend pas de paramètre,
retourne false ou l'array des lignes de la table produits de la base de
données, avec l'attribut image.
        suite à l'interrogation de la bdd
         */
    public function getAllProductsWithImages()
    {
         $this->db=new DBController;
         if($this->db->openConnection())
         {
            $query="select
products.id,products.name,products.price,products.quantity,categories.name
as 'category',products.image, products.description from products,categories
where products.categoryid=categories.id;";
            return $this->db->select($query);
         }
         else
```

```
            {
               echo "Error in Database Connection";
               return false;
            }
    }

        /*
        Methode getCategoryProducts(), prend en paramètre l'id de la
catégorie ciblée, retourne false ou l'array des lignes de la table
catégorie de la base de données, qui ont pour categorieid le paramètre $id.
        suite à l'interrogation de la bdd
         */
    public function getCategoryProducts($id)
    {
         $this->db=new DBController;
         if($this->db->openConnection())
         {
            $query="select
products.id,products.name,price,quantity,categories.name as
'category',image from products,categories where
products.categoryid=categories.id and categories.id=$id;";
            return $this->db->select($query);
         }
         else
         {
            echo "Error in Database Connection";
            return false;
         }
    }


}
```

II. Models :

1. Category :

```
/* Nous construisons ici le modèle d'un objet catégorie, il sera
caractérisé par deux attributs, son id et son nom,
   le constructeur par defaut n'est pas modifié car il fait le boulot par
rapport à notre besoin.

 */
class Category
{
    public $id;
    public $name;

}

?>
```

2. product :

```
/* Nous construisons ici le modèle d'un objet produit, il sera caractérisé
par 7 attributs, l'id, le nom commercial,
   une description courte de 500 caractères, le prix, la quantité
disponible en stock, l'image ( voir le commentaire associé),
```

```
   et l'id de la catégorie à laquelle il appartient.
   le constructeur par defaut n'est pas modifié car il fait le boulot par
rapport à notre besoin.

 */

class Product
{
    public $id;
    public $name;
    public $description;
    public $price;
    public $quantity;
    /* Dans une première version, dans la bdd, l'image a été sockée sous
forme binaire ( blob ), ceci s'est avéré
       contraire aux bonnes pratiques et source de bugs (en fonction de la
nature de l'extension, problèmes de compatibilité
       avec le navigateur). Le correctif est : de stocker le chemin vers la
photo. Ici le dossier des images de produits
       est stockée dans les assets/img/ . En conclusion, l'attribut $image
est désormais une chaine de maximum 500 charactères
       spécifiant le chemin vers l'image du produit.
    */
    public $image;
    public $categoryid;

}

?>
```

3. role :

```
<?php

/* Nous construisons ici le modèle d'un objet role, il sera caractérisé par
deux attributs, son id et son nom,
   le constructeur par defaut n'est pas modifié car il fait le boulot par
rapport à notre besoin.

 */

class Role
{
    public $id;
    public $name;

}

?>
```

4. user :

```
<?php

/* Nous construisons ici le modèle d'un objet utilisateur, il sera
caractérisé par cinq attributs, son id, son nom,
   son email, son mot de passe, et le rôle qu'il joue ( dans notre cas :
client normal ou administrateur).
   le constructeur par defaut n'est pas modifié car il fait le boulot par
```

```
rapport à notre besoin.

 */

class User
{
    public $id;
    public $name;
    public $email;
    public $password;
    public $roleid;


}

?>
```

III. Views:

1. Admin :
   a- crud.php :

```
<?php

// Exécuter un session_start() pour avoir accès à la variable $_SESSION
dans tous les cas
session_start();

 /*

 Ce code permet l'affichage et le fonctionnement de la page CRUD de
l'administrateur,
 alors tout naturellement on commence par vérifier qu'il y a bien un usager
actif,
 et qu'il s'agit bien d'un administrateur

 */

// S'il n'y a pas d'utilisateur actif ( compte connecté)
if (!isset($_SESSION["userRole"])) {

    // il est redirigé vers la page de logging
    header("location:../Auth/login.php ");

//  Et s'il y en a un,
} else {

    // on vérifie que c'est bel et bien un administrateur
    if ($_SESSION["userRole"] != "Admin") {

        // S'il ne l'est pas, on le redirige vers la page d'accueil des
clients connectés
        header("location:../Client/logged_client.php ");
    }
}


/* Après ces vérifications, nous sommes assurés que nous avons un
administrateur connecté.
   Alors, il faut lui afficher la page avec les outils appropriés.
*/
```

```php
// Nous récupérons nos outils ( classes et méthodes ..)
require("../../tbs_class.php");
require_once '../../Controllers/ProductController.php';
require_once '../../Models/product.php';

/*
Dans cette page l'administrateur a la liste complète des produits affichée
au milieu, avec un bouton lui permettant
d'éliminer un produit à la fois ( effacer la ligne correspondante au
produit choisi de la base de données).
*/

// On instancie un objet de la classe ProductController
$productController = new ProductController;

// On récupère la liste des produits diponibles en utilisant le Getter des
produits de l'objet $productController
$produits = $productController->getAllProducts();

// On déclare un message un vide que l'on va adapter en fonction du
déroulement de nos opérations
$deleteMsg = "";

// S'il y a réception d'une requête de suppression du formulaire
if (isset($_POST["delete"])) {

    // et si cette requête n'est pas vide
    if (!empty($_POST["productId"])) {

        // la méthode deleteProducte() offerte par le controleur de produit
permet d'effacer la ligne du produit dont l'id a été
        // posté par le formulaire,de la base de données.
        if ($productController->deleteProduct($_POST["productId"])) {

            // Si la méthode deleteProduct() est executée avec succès:
            // On affecte une valeur de succès de l'opération au message à
afficher
            $deleteMsg = "Produit supprimé";

            // On re-récupère la liste des produits, pour que le produit
supprimé ne soit plus affiché
            $produits = $productController->getAllProducts();
        }
    }
}


// on récupère le nom et le rôle de l'utilisateur
$utilisateur = $_SESSION["userName"];
$role = $_SESSION["userRole"];


// On affiche le tout grâce à tbs:
$tbs = new clsTinyButStrong ;
$tbs->LoadTemplate("Views/Admin/crud.html");
$tbs->MergeBlock('produit', $produits);
$tbs->MergeField('deleteMsg', $deleteMsg);
$tbs->MergeField('utilisateur', $utilisateur);
$tbs->MergeField('role', $role);
$tbs->Show();
```

b- Crud2.php :

```php
<?php
// Exécuter un session_start() pour avoir accès à la variable $_SESSION
dans tous les cas
session_start();

/*

 Ce code permet l'affichage et le fonctionnement de la page CRUD de
l'administrateur,
 alors tout naturellement on commence par vérifier qu'il y a bien un usager
actif,
 et qu'il s'agit bien d'un administrateur

 */

// S'il n'y a pas d'utilisateur actif (compte connecté)
if (!isset($_SESSION["userRole"])) {

    // Il est redirigé vers la page de logging
    header("location:../Auth/login.php ");

// Et s'il y en a un
} else {

    // Vérifier qu'il s'agisse d'un administrateur
    if ($_SESSION["userRole"] != "Admin") {

        // S'il ne l'est pas, il est redirigé vers la page d'accueil client
connecté
        header("location:../Client/logged_client.php ");
    }
}

/* Après ces vérifications, nous sommes assurés que nous avons un
administrateur connecté.
   Alors, il faut lui afficher la page avec les outils appropriés.
*/

// Nous récupérons nos outils (classes et méthodes ..)
require("../../tbs_class.php");
require_once '../../Controllers/ProductController.php';
require_once '../../Models/product.php';

/*

Dans cette page l'administrateur a un formulaire d'ajout de produit affiché
au milieu de la page, lui donnant la possibilité
de définir une nouvelle ligne dans la base de données, correspondant a à
nouveau produit. Pour lui permettre de réaliser
cette opération :

*/

// On instancie un objet de la classe ProductController
$productController = new ProductController;

// On récupère la liste des catégories, pour les proposer dans une liste
d'options lors de la définition du nouveau produit
$categories = $productController->getCategories();
```

```php
// On déclare un message d'erreur vide que l'on va adapter en fonction du déroulement des opérations
$errMsg = "";


// Vérifier que les valeurs transmises par le "post" sont bien renseignées
if (isset($_POST['name']) && isset($_POST['description']) &&
isset($_POST['price']) && isset($_POST['quantity']) &&
isset($_FILES["image"])) {


    if (!empty($_POST['name']) && !empty($_POST['description']) &&
!empty($_POST['price']) && !empty($_POST['quantity'])) {


        // A ce stade nous disposons d'un ensemble de valeurs que l'on doit affecter aux attributs d'un nouvel objet produit

        // Commencer par instancier un nouvel objet de la classe Product
        $product = new Product;

        // Et affecter les valeurs transmises par le post aux attributs de ce nouvel objet
        $product->name = $_POST['name'];
        $product->description = $_POST['description'];
        $product->price = $_POST['price'];
        $product->quantity = $_POST['quantity'];
        $product->categoryid = $_POST['category'];

        /*
        Pour l'attribut image:
        Nous avons vu lors de la construction de la base de données, que l'attribut image est un "chemin d'accès" (bonnes pratiques).
        Dans le formulaire d'ajout de nouveau produit, l'administrateur récupère un fichier image d'un repertoire quelconque, il faut pouvoir :
        - stocker le contenu de l'image spécifiée, dans le repertoire qu'on a choisi pour les images de nos produits.
        - construire le chemin d'accès à cette image stockée, pour le fournir en valeur pour l'attribut image dans la bdd

        Les pas suivants permettent de réaliser cette opération :

        */

        // Concaténer l'adresse du repertoire d'images produits, choisi pour l'application, avec le nom du fichier récupéré,
        // le résultat est une chaine de caractères représentant le chemin d'accès à cette image dans notre application
        $location = "../../Views/assets/img/produits/".
$_FILES["image"]["name"];

        // Placer le contenu de l'image dans cette nouvelle adresse, si l'opération est réussie :
        if (move_uploaded_file($_FILES["image"]["tmp_name"], $location)) {

            // on affecte cette valeur (adresse) à l'attribut image de notre objet $product
            $product->image = $location;
            /*
             On fait appel à la méthode addProduct de la classe productController, et on l'alimente avec
```

```
             le paramètre $product, Si l'opération est réussie (création
d'une nouvelle ligne d'un nouveau produit,
             dans la table "produits" de la base de données :
             */
            if ($productController->addProduct($product)) {

                // On redirige vers la vue crud.php, ou l'administrateur
pourra constater une nouvelle ligne produit dans l'affichage
                header("location: crud.php");

            // si la methode addProduct() rencontre un problème:
            } else {

                // On adapte le message d'erreur
                $errMsg = "L'ajout n'a pas pu aboutir, merci de réessayer";
            }

        // Si le contenu de l'image téléchargée n'a pas pu être stockée à
l'adresse spécifiée
        } else {

            // On adapte le message d'erreur
            $errMsg = "Erreur lors du chargement";
        }

    // Si l'un des champs du formulaire n'a pas été correctement rempli,
adapter le message d'erreur en conséquence
    } else {
        $errMsg = "Merci de remplir tous les champs";
    }
}


// Pour le message d'accueil et le statut, on récupère les valeurs pour
l'administrateur connecté
$utilisateur = $_SESSION["userName"];
$role = $_SESSION["userRole"];


// on utilise tbs pour afficher (commentaires plus détaillés dans la vue
d'accueil par exemple)
$tbs = new clsTinyButStrong ;
$tbs->LoadTemplate("Views/Admin/crud2.html");

// Options de la catégorie, dans le menu select déroulant, l'administrateur
a juste à choisir le nom de la catégorie dans la liste, et
// ça sera l'id de la catégorie qui sera transmis dans le formulaire.
$tbs->MergeBlock('categorie', $categories);


$tbs->MergeField('errMsg', $errMsg);
$tbs->MergeField('utilisateur', $utilisateur);
$tbs->MergeField('role', $role);
$tbs->Show();
```

2. Auth :

a- login.php :

```
<?php
```

```php
// Boite à outils: récupérer nos outils encapsulés ( classes, methodes associées..)
require("../../tbs_class.php");
require_once '../../Models/user.php';
require_once '../../Controllers/AuthController.php';

// déclarer un message d'erreur vide, que l'on va affiner en fonction de l'execution de notre logique
$errMsg="";


// quand le formulaire est utilisé, on vérifie que les champs email et password de $_POST ont été initiés
if(isset($_POST['email']) && isset($_POST['password']))
{
    // dans ce cas , on vérifie qu'ils n'ont pas été initiés avec des champs vides
    if(!empty($_POST['email']) && !empty($_POST['password']) )
    {
        // Si les deux conditions précédentes sont remplies:
        // on sort la boite à outils, on instancie un objet de la classe User
        $user=new User;
        // ainsi qu'un objet AuthController
        $auth=new AuthController;
        // on affecte la valeur email postée, à l'attribut email de l'objet User
        $user->email=$_POST['email'];
        // on affecte la valeur password postée, à l'attribut password de l'objet User
        $user->password=$_POST['password'];
        // on utilise la méthode login() de l'objet Auth, avec cet $user en paramètre, et on vérifie le retour
        if(!$auth->login($user))
        {
            if(!isset($_SESSION["userId"]))
            {
                session_start();
            }

            // si la méthode login() renvoie false ( echec de la connexion), on récupère le message d'erreur
            // qu'elle a stockée dans $_SESSION et on l'affecte à notre variable message errMsg
            $errMsg=$_SESSION["errMsg"];
        }

        // Et si la méthode login() renvoie true, la connexion a eu lieu
        else
        {
            if(!isset($_SESSION["userId"]))
            {
                session_start();
            }

            // on redirige l'usager en fonction de son rôle:

            if($_SESSION["userRole"]=="Admin")
            {
                // s'il est "admin" on le dirige vers la page ou il peut créer lire mettre à jour et supprimer des données
```

```
                header("location: ../Admin/crud.php");
            }
            else
            {
                // si l'usager est juste un client, il est redirigé vers la
page d'accueil des clients connectés
                header("location: ../Client/logged_client.php");
            }

        }


    }

    // si le champ email ou password n'a pas été rempli lors de l'envoi de
la requête, un message adéquat est affecté
    else
    {
        $errMsg="Merci de remplir tous les champs";
    }
}


// utiliser tbs pour récupérer et afficher le gabarit:

// Instancier un objet de la classe tbs
$tbs = new clsTinyButStrong ;

// charger le template html
$tbs->LoadTemplate("Views/Auth/login.html");

// injecter le message d'erreur (s'il y en a un ), à partir de notre
variable $errMsg
$tbs->MergeField('errMsg', $errMsg);

// afficher le résultat
$tbs->Show();
```

b- register.php :

```
<?php

// Récupérer nos outils
require("../../tbs_class.php");
require_once '../../Models/user.php';
require_once '../../Controllers/AuthController.php';

// Déclarer un message d'erreur vide qui sera affiné en fonction du
déroulement
$errMsg="";

// Exécuter un session_start() s'il n'y a pas de session d'utilisateur
actif en cours
if(!isset($_SESSION["userId"]))
{
  session_start();
}
```

```php
// Vérifier si le formulaire a "posté" des éléments d'enregistrement d'un
usager
if (isset($_POST['email']) && isset($_POST['password']) &&
isset($_POST['name']))
{
  // Si c'est le cas, vérifier que ces éléments sont bien renseignés (pas
de champ de vide)
  if (!empty($_POST['email']) && !empty($_POST['password']) &&
!empty($_POST['name']))
  {
    // Si on a bien récupére un email, un password et un nom, on procède à
la création d'un nouveau compte:

    // Instancier un objet de la classe User ( voir les commentaires
associés à chaque classe et ses méthodes )
    $user=new User;

    // Instancier un objet de la classe AuthController ( voir les
commentaires associés à chaque classe et ses méthodes )
    $auth=new AuthController;

    // Affecter les valeurs récupérées du post, aux attributs de l'objet
user
    $user->name=$_POST['name'];
    $user->email=$_POST['email'];
    $user->password=$_POST['password'];

    /* exécuter la méthode register() de l'objet $auth, avec comme
paramètre l'objet $user confectionné avec
       les valeurs récupérées du post.
       Si la méthode est executée avec succès, un nouvel utilisateur existe
désormais dans la base de  données.
       On redirige en conséquence l'utilisateur vers la page de logging.
    */
    if($auth->register($user))
    {
      header("location: ../Auth/login.php");
    }
    /* Si la methode register() n'est pas exécutée avec succès, elle aura
inscrit un message d'erreur dans la session,
       on affecte ce message d'erreur à la variable $errMsg à afficher dans
la vue "register"
    */
    else
    {
      $errMsg=$_SESSION["errMsg"];
    }

  }
  // Si l'une des valeurs transmises par le post est vide, on adapte le
message d'erreur pour l'afficher dans la vue "register"
  else
  {
    $errMsg="Merci de remplir tous les champs";
  }

}

// utiliser tbs pour récupérer et afficher le gabarit

//Instancier un objet tbs
```

```php
$tbs = new clsTinyButStrong ;

// Charger le template html "register"
$tbs->LoadTemplate("Views/Auth/register.html");

// Injecter la variable $errMsg dans la vue
$tbs->MergeField('errMsg', $errMsg);

// Afficher le resultat
$tbs->Show();
```

3. Client :
   a- home.php :

```php
<?php
// récupérer nos outils encapsulés
require("../../tbs_class.php");
require_once '../../Controllers/ProductController.php';
require_once '../../Models/product.php';
// executer un session_start() afin d'avoir accès aux variables de la session
session_start();


/* Nous gérons ici sur la page d'accueil du site, qui doit s'afficher en dehors d'un client ou un administrateur loggé,
   ou si on y est redirigé suite à une demande de déconnexion. */


/*nous commençons par vérifier qu'il s'agit pas d'une redirection suite demande de deconnexion, si c'est le cas, on
   détruit la session (effacer les valeurs de variables de l'usager en cour) */

if(isset($_GET["logout"]))
{
    session_start();
    session_destroy();
}


// S'il n'y a pas eu redirection suite à une demande de logout, on vérifie si un usager est loggé :
if (isset($_SESSION["userRole"])) {

    // Si l'usager est un client, on le dirige vers la page d'accueil client loggé
    if($_SESSION["userRole"]=="Client"){
        header("location:../../Views/Client/logged_client.php ");
    }

    // Si l'usager est un admin, on le dirige vers la page d'administration
    if($_SESSION["userRole"]=="Admin"){
        header("location:../../Views/Admin/crud.php ");
    }

}

// Si aucun usager n'est loggé, on affiche notre page d'accueil, pour celà:
```

```php
// Il faut récupérer de la base de données la liste des produits et leurs caractéristiques, ainsi que la liste des catégories:
$productController = new ProductController;
$categories = $productController->getCategories();
$produits = $productController->getAllProductsWithImages();


// tbs est la pour charger le gabarit et y injecter les données récupérées de la base de données:

// Instancier un objet tbs
$tbs = new clsTinyButStrong ;

// Charger le template html
$tbs->LoadTemplate("Views/Client/home.html");

// Injecter toutes les catégories dans la vue ( onglet catégories, en haut à gauche )
$tbs->MergeBlock('categorie', $categories);

// Injecter tous les produits disponibles dans la vue ( espace contenu, au centre )
$tbs->MergeBlock('produit', $produits);

// Et... afficher le résultat, magique tbs !
$tbs->Show();
```

b- logged_client.php :

```php
<?php

// récupérer nos outils encapsulés
require("../../tbs_class.php");
require_once '../../Controllers/ProductController.php';
require_once '../../Models/product.php';

// executer la commande session_start() afin d'avoir accès aux variables de la session
session_start();


// Nous gérons ici sur la page d'un client loggé:

// nous commençons par vérifier qu'il y a bien un usager loggé ( notons que l'admin a accès à cette page même s'il n'est pas client)

// si l'usager n'est pas loggé :
if (!isset($_SESSION["userRole"])) {

    // On le dirige vers la page d'accueil standard

    header("location:../../Views/Client/home.php ");


}

// Si l'usager est loggé, on affiche la page complétée par:
```

```php
// 1-Les données sur les produits
$productController = new ProductController;
$categories = $productController->getCategories();
$produits = $productController->getAllProductsWithImages();

//2-Les données sur le client
$utilisateur = $_SESSION["userName"];
$role = $_SESSION["userRole"];

// tbs est la pour charger le gabarit et y injecter les données récupérées
de la base de données:

// Instancier un objet tbs
$tbs = new clsTinyButStrong ;

// Charger le template html
$tbs->LoadTemplate("Views/Client/logged_client.html");

// Injecter toutes les catégories dans la vue ( onglet catégories en haut à
gauche )
$tbs->MergeBlock('categorie', $categories);

// Injecter tous les produits disponibles dans la vue ( espece contenu au
centre )
$tbs->MergeBlock('produit', $produits);

// Injecter le nom de l utilisateur dans le message de bienvenue ( visible
après appui sur l'icone image utilisateur en haut à droite )
$tbs->MergeField('utilisateur', $utilisateur);

// Injecter le role de l'utilisateur dans le message de "statut de
l'utilisateur" ( idem )
$tbs->MergeField('role', $role);

// Et... afficher le résultat, magique tbs !
$tbs->Show();
```

IV. Script SQL Base de données :

```
-- phpMyAdmin SQL Dump

-- version 5.2.0

-- https://www.phpmyadmin.net/

--

-- Hôte : mysql02.univ-lyon2.fr

-- Généré le : lun. 17 avr. 2023 à 23:31

-- Version du serveur : 5.5.64-MariaDB

-- Version de PHP : 8.2.0alpha1
```

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Base de données : `php_memansour`
--

-- --------------------------------------------------------

--
-- Structure de la table `categories`
--

CREATE TABLE `categories` (
  `id` int(11) NOT NULL,
  `name` varchar(100) CHARACTER SET utf8 NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Déchargement des données de la table `categories`
--

INSERT INTO `categories` (`id`, `name`) VALUES
```

```
(1, 'Data Science'),

(2, 'Big Data'),

(3, 'Machine Learning'),

(4, 'Asimov');


-- --------------------------------------------------------


--

-- Structure de la table `products`

--


CREATE TABLE `products` (

  `id` int(11) NOT NULL,

  `name` varchar(220) CHARACTER SET utf8 NOT NULL,

  `description` varchar(499) CHARACTER SET utf8 NOT NULL,

  `price` varchar(10) CHARACTER SET utf8 NOT NULL,

  `quantity` int(11) NOT NULL,

  `image` varchar(500) CHARACTER SET utf8 NOT NULL,

  `categoryid` int(11) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;


--

-- Déchargement des données de la table `products`

--


INSERT INTO `products` (`id`, `name`, `description`, `price`,
`quantity`, `image`, `categoryid`) VALUES

(1, 'BI et Big data', 'L\'entreposage de données et l\'analyse en ligne
(On-Line Analytical Processing _OLAP) se sont imposés comme des outils
fondamentaux de l\'informatique décisionnelle (Business Intelligence -
BI). Avec l\'avènement des mégadonnées (big data) caractérisées par leur
très grand volume, leur vélocité et leur variété, et des technologies et
infrastructures émergentes (NoSQL, Cloud, Hadoop...), ces outils
d\'analyse et d\'aide à la décision sont confrontés à de nouveaux défis
```

```
scientifiques.', '60.00', 22,
'../../Views/assets/img/produits/BIetBigdata.webp', 2),

(2, 'Python pour la Data Science', 'Pour de nombreux chercheurs, Python
constitue l\'outil de prédilection en raison de ses riches librairies de
fonctions de traitement et d\'analyse de données. ', '39.95', 3,
'../../Views/assets/img/produits/pythonpourladatascience.webp', 1),

(3, 'Big Data et Machine Learning', 'Cet ouvrage s\'adresse à tous ceux
qui cherchent à tirer parti de  l\'énorme potentiel des « technologies
Big Data », qu\'ils soient data  scientists, DSI, chefs de projets ou
spécialistes métier. ', '24.99', 5,
'../../Views/assets/img/produits/bigdataetmachinelearning.webp', 3),

(4, 'Fondation', 'En ce début de treizième millénaire, l\'Empire n\'a
jamais été aussi puissant, aussi étendu à travers toute la galaxie. Et
c\'est dans sa capitale, Trantor, que Hari Seldon invente la
psychohistoire, une science nouvelle permettant de prédire l\'avenir.',
'7.99', 1, '../../Views/assets/img/produits/fondation.webp', 4),

(5, 'Extraction et Gestion de la Connaissance', 'La sélection
d\'articles publiés dans le présent recueil constitue les actes des 23e
Journées Internationales Francophones Extraction et Gestion des
Connaissances (EGC 2023) qui se sont déroulées à l\'Université Lumière
Lyon 2 du 16 janvier au 20 janvier 2023. ', '80.00', 3,
'../../Views/assets/img/produits/extractionetgestiondesconnaissances.web
p', 2),

(6, 'Analytics', 'By leveraging big data & analytics, businesses create
the potential to better understand, manage, and strategically exploiting
the complex dynamics of customer behavior', '32.99', 5,
'../../Views/assets/img/produits/analytics.jpg', 2);



-- --------------------------------------------------------



--

-- Structure de la table `roles`

--



CREATE TABLE `roles` (

  `id` int(11) NOT NULL,

  `name` varchar(50) CHARACTER SET utf8 NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;



--

-- Déchargement des données de la table `roles`
```

```
--

INSERT INTO `roles` (`id`, `name`) VALUES
(1, 'Admin'),
(2, 'Client');

-- --------------------------------------------------------

--
-- Structure de la table `users`
--

CREATE TABLE `users` (
  `id` int(11) NOT NULL,
  `name` varchar(50) CHARACTER SET utf8 NOT NULL,
  `email` varchar(100) CHARACTER SET utf8 NOT NULL,
  `password` varchar(100) CHARACTER SET utf8 NOT NULL,
  `roleid` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Déchargement des données de la table `users`
--

INSERT INTO `users` (`id`, `name`, `email`, `password`, `roleid`) VALUES
(1, 'mehdi mansour', 'mehdi.mansour@univ-lyon2.fr', 'password', 1),
(2, 'pierre dupont', 'pierredupont@gmail.com', 'password', 2),
(3, 'admin', 'admin@gmail.com', 'password', 1),
(4, 'client1', 'client1@gmail.com', 'password', 2),
(5, 'julio', 'julio@gmail.com', 'password', 2),
```

```
(6, 'elon', 'elon@gmail.com', 'password', 2),
(7, 'titi', 'titi@gmail.com', 'password', 2),
(8, 'Rasmus Lerdorf', 'super_admin@gmail.com', 'password', 1),
(9, 'Pierre Dupont', 'client_lambda@gmail.com', 'password', 2);

--
-- Index pour les tables déchargées
--

--
-- Index pour la table `categories`
--
ALTER TABLE `categories`
  ADD PRIMARY KEY (`id`);

--
-- Index pour la table `products`
--
ALTER TABLE `products`
  ADD PRIMARY KEY (`id`),
  ADD KEY `categoryid` (`categoryid`);

--
-- Index pour la table `roles`
--
ALTER TABLE `roles`
  ADD PRIMARY KEY (`id`);

--
-- Index pour la table `users`
```

```
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD KEY `fk_users_roleid` (`roleid`);

--
-- AUTO_INCREMENT pour les tables déchargées
--

--
-- AUTO_INCREMENT pour la table `categories`
--
ALTER TABLE `categories`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;

--
-- AUTO_INCREMENT pour la table `products`
--
ALTER TABLE `products`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=13;

--
-- AUTO_INCREMENT pour la table `roles`
--
ALTER TABLE `roles`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=3;

--
-- AUTO_INCREMENT pour la table `users`
--
```

```
ALTER TABLE `users`

  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;



--

-- Contraintes pour les tables déchargées

--



--

-- Contraintes pour la table `products`

--

ALTER TABLE `products`

  ADD CONSTRAINT `products_ibfk_1` FOREIGN KEY (`categoryid`) REFERENCES
`categories` (`id`);



--

-- Contraintes pour la table `users`

--

ALTER TABLE `users`

  ADD CONSTRAINT `fk_users_roleid` FOREIGN KEY (`roleid`) REFERENCES
`roles` (`id`);

COMMIT;



/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;

/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;

/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```