



Optimisation fonctionnelle du gestionnaire

Sommaire

Introduction.....	3
I. Affichage des informations de la base pour chaque image	4
A. Gérer les mises à jour des images en fonction de leur statut	6
B. Finalisation : ajout des informations sur la page de contenu <i>front</i>	11
II. La gestion des erreurs	11
A. Inclusion du fichier <i>process_image.php</i> dans le fichier <i>admin.php</i>	12
B. Modification des méthodes de la classe <i>Image</i>	12
C. Modification du fichier <i>process_image.php</i>	13
D. Intervention sur la table : index unique pour le champ <i>filename</i>	14
III. Tests de vérification, erreurs volontaires	14
A. Première erreur volontaire	14
B. Deuxième erreur volontaire.....	15
C. Troisième erreur volontaire	15

Crédits des illustrations : © Fotolia

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz



Introduction

Ainsi nous allons dans l'ordre :

- afficher les informations de la base pour chaque image ;
- alterner nouvelle insertion *INSERT* et mise à jour *UPDATE* en fonction du statut de chaque image dans la base ;
- gérer l'affichage des messages d'erreur ;
- puis développer une interface d'*upload*, permettant le chargement de fichiers sur le site.

I. Affichage des informations de la base pour chaque image

La problématique consiste ici à associer la restitution des fichiers du répertoire avec la restitution des informations de la base. **Les fichiers et la base de données sont deux environnements hétérogènes** et il sera nécessaire de vérifier à chaque fois que les fichiers et les données peuvent être associés et qu'ils existent.



Les termes « contenu », « information » et « donnée » sont synonymes et servent à évoquer ce qui est enregistré dans la base de données.

Pour vérifier le fait que les fichiers d'une part et les données de la base d'autre part peuvent être associés, nous devons utiliser une *valeur commune qui nous servira de clef unique*. Il se trouve que le nom d'un fichier est unique pour un répertoire donné. En effet, il ne peut pas techniquement y avoir deux fichiers portant le même nom de fichier dans un même répertoire.

Le nom d'un fichier sert donc immanquablement à identifier celui-ci (et c'est bien évidemment cette règle qui permet la gestion des fichiers quel que soit l'environnement d'exploitation).

Le nom du fichier en tant qu'identifiant du fichier pourra de ce fait être utilisé dans la base de données pour associer les données enregistrées au fichier en question.

C'est le cas actuellement de notre développement : la base de données contient des données (titre et description) qui sont finalement dédiées au fichier **image1.jpg** car la table contient le champ (colonne) *filename* permettant d'identifier à quel fichier ces données se rapportent.

Tableau n°1 Table image

ID	TITLE	DESCRIPTION	FILENAME
1	test titre image 1	test description image 1	image1.jpg
2	test titre image 2	test description image 2	image2.jpg

La table contient le champ *id* qui, classiquement, sert de clé primaire. Mais nous ne pouvons pas nous en servir pour associer le contenu et les fichiers.

Ainsi pour restituer le contenu, nous allons modifier la classe *Image* en ajoutant une méthode dédiée à la récupération des contenus de la base ce qui nécessitera par la suite de mettre à jour la méthode *getImages* ainsi que le fichier de restitution **admin.php**.

Nous créons la méthode *getImageData* qui a pour objectif de récupérer au moyen d'un *SELECT* le contenu de la table *image* grâce à l'identifiant de l'image enregistré dans le champ *filename*.

Notons que le nom de la méthode *getImageData* est calqué sur le principe de nommage évoqué précédemment. Le terme *data* indique par convention qu'il s'agit d'informations issues d'une base de données.

Le nom *getImageData* signifie donc : **obtenir les informations de la base pour une image**. Si nous avons créé une méthode permettant de récupérer les informations pour plusieurs images, nous aurions ajouté un *S* à *image* et obtenu ainsi le nom *getImagesData*.

Revenons à la méthode *getImageData*, qui contiendra la simple requête SQL suivante :

```
SELECT id, title, description, filename
FROM image WHERE filename = '' . $filename .'
```

Fig. 1

Nous intégrerons cette requête, **via la méthode *getImageData***, dans la méthode *getImages*. Ainsi à chaque fois qu'une image sera ajoutée au tableau *\$images* (qui liste tous les fichiers) nous effectuerons cette requête (le nom de l'image étant affecté à *\$filename*).

Nous pourrions ainsi récupérer les informations de la base et les ajouter comme contenu supplémentaire dans le tableau *\$images* final.

Voici donc le code de la méthode qui nous permet de retourner le tableau *\$image_data* contenant donc les informations de la table *image* :

```
public function getImageData($filename)
{
    $mysqli = new mysqli('localhost', 'root', '', 'projet_image');
    $mysqli->set_charset("utf8"); // encodage utf8
    /* Vérification de la connexion */
    if ($mysqli->connect_errno) {
        printf("Echec de la connexion: %s\n", $mysqli->connect_error);
        exit();
    }
    $result = $mysqli->query('SELECT id, title, description, filename
FROM image WHERE filename = '' . $filename .'
```

Fig. 2

Bien que la requête SQL soit simple, nous serons vigilants concernant la concaténation et les doubles *quotes* utilisées autour de la variable *\$filename* qui indiquent que le champ *filename* est de type *VARCHAR*.

Comme convenu, nous intégrons maintenant cette méthode dans la méthode *getImages* qui parcourt le répertoire *image* et retourne la liste des fichiers images.

A. Gérer les mises à jour des images en fonction de leur statut

Nous modifions donc la méthode *getImages* :

```
public function getImages($image_dir)
{
    //iterator
    $i=0;
    if ($handle = opendir($image_dir))
    {
        while (false !== ($entry = readdir($handle)))
        {
            if ($entry != "." && $entry != "..")
            {
                $i++;
                $images[$i]['filename'] = $entry;
                // utilisation de $this pour appeler la méthode getImageData
                $image_data = $this->getImageData($entry);
                $images[$i]['title'] = $image_data['title'] ;
                $images[$i]['description'] = $image_data['description'] ;
            }
        }
        closedir($handle);
        return $images ;
    }
}
```

Fig.3

Les deux modifications effectuées sont les suivantes :

1. Intégration de la méthode *getImageData* au moyen de la pseudo-variable *\$this*

```
$image_data = $this->getImageData($ent
```

Fig.4

Le résultat retourné par cette méthode est un tableau que nous affectons à une variable que nous appelons *\$image_data* (qui est donc un tableau). Nous récupérons dans ce tableau les valeurs associées aux index *title* et *description* et les ajoutons au tableau *\$images* destiné à être retourné par la méthode *getImages*.

2. Ajout d'un itérateur *\$i* permettant de générer le tableau *\$images*

Il s'agit d'une méthode classique, déjà abordée dans les cours précédents, qui permet de créer des index pour les tableaux associatifs. La variable *\$i* (i pour itérateur) sera incrémentée de 1 à chaque tour de boucle (itération).

Il ne reste plus qu'à modifier le fichier final **admin.php** qui sert à afficher les informations générées par la méthode *getImages*.

Puisque nous récupérons un tableau de données, nous allons tester les valeurs avec un *var_dump*. Nous vérifions que nous récupérons correctement les deux fichiers images présents dans le répertoire et que ces deux images sont bien référencées dans la table *image*.

```
array
1 =>
  array
    'filename' => string 'image1.jpg' (length=10)
    'title' => string 'test titre image 1' (length=18)
    'description' => string 'test description image 1' (length=24)
2 =>
  array
    'filename' => string 'image2.jpg' (length=10)
    'title' => string 'test titre image 2' (length=18)
    'description' => string 'test description image 2' (length=24)
```

Fig.5

Nous constatons que les informations sont correctes et pouvons maintenant modifier le fichier **admin.php** de manière à afficher ces informations.

La modification est simple et consiste à :

- récupérer le tableau associatif qu'est devenue la variable *\$images* (qui était donc initialement un tableau numérique simple) ;
- afficher dans la zone de texte *textarea* ainsi que dans les champs de formulaire au moyen de l'élément HTML *value*, les titres et les descriptions.

```
<?php
require('config.php');
require('Image.php');
$image = new Image();
$images = $image->getImages(IMAGE_DIR_PATH);
?>

<h1><?php echo WEB_TITLE ?></h1>
<ul>
<?php foreach ($images as $image): ?>
<li>
<form method="post" action="process_image.php">
<p>Titre : <input type="text" name="title" value="<?php echo
$image['title'] ?>" /></p>
<input type="hidden" name="filename" value="<?php echo
$image['filename'] ?>" />
<p>Description<br> <textarea name="descr" cols="50" rows="2"><?php
echo $image['description'] ?></textarea></p>
<p><input type="submit" name="formImageSubmit" value="validez" /></p>
</form>
</li>
<?php endforeach ?>
</ul>
```

Fig.6

La restitution dans la page web est donc la suivante :

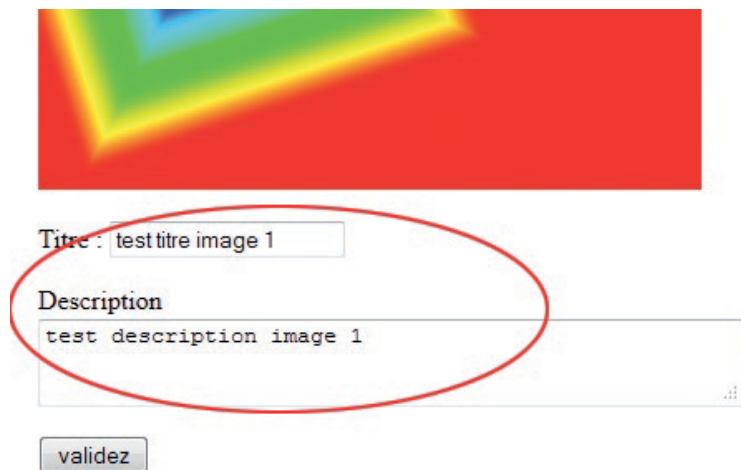


Fig.7 Le titre et la description sont dorénavant restitués dans le formulaire

Le développement semble correct mais nous devons vérifier s'il fonctionne lorsque nous ajoutons une troisième image dans le répertoire « *images* ». Nous ajoutons donc « à la main » (c'est-à-dire sans interface web) un nouveau fichier que nous nommons **image3.jpg**.

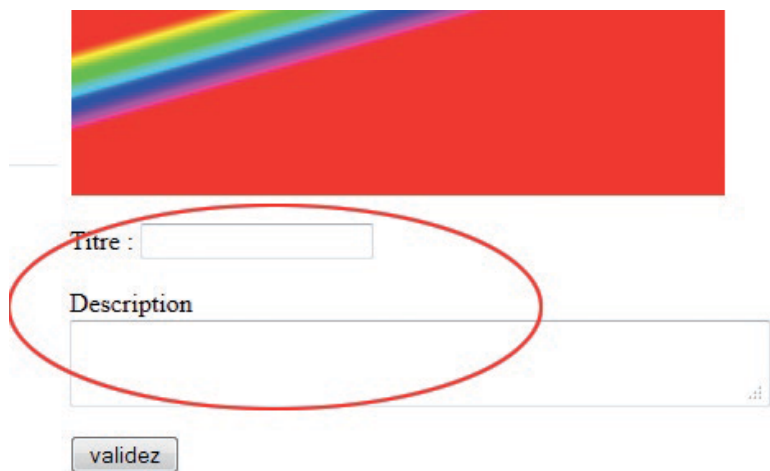


Fig.8

Nous constatons que cette troisième image apparaît bien dans la page web mais que les titres et descriptions sont vides puisque nous n'avons pas encore saisi le contenu.

Il peut être intéressant de se poser la question de savoir pourquoi aucune erreur n'est générée.

En effet, le script de la page **admin.php** utilise directement les variables issues du tableau comme suit :

```
value="<?php echo $image['title'] ?>" />
```

Fig.9

En toute logique, la variable n'existant pas pour cette troisième image, une erreur PHP devrait apparaître.

Afin de comprendre pourquoi ce n'est pas le cas, regardons à nouveau ce qu'affiche le `var_dump`.

Le `var_dump` avec maintenant trois images dont deux seulement sont référencées dans la table `image` est le suivant :

```
array
1 =>
  array
    'filename' => string 'image1.jpg' (length=10)
    'title' => string 'test titre image 1' (length=18)
    'description' => string 'test description image 1' (length=24)
2 =>
  array
    'filename' => string 'image2.jpg' (length=10)
    'title' => string 'test titre image 2' (length=18)
    'description' => string 'test description image 2' (length=24)
3 =>
  array
    'filename' => string 'image3.jpg' (length=10)
    'title' => null
    'description' => null
```

Fig. 10

Nous voyons bien que les trois noms de fichier des images sont présents ainsi que les informations issues de la base de données pour les deux premières et que ces valeurs sont naturellement des chaînes de caractères (*string*). En revanche, et c'est là tout l'intérêt du `var_dump`, nous constatons que les clefs «titre» et «description» correspondent à des valeurs de type *null* pour la troisième image. Une **valeur** de type *null* (ou *NULL*) n'existe tout simplement pas, alors que la **variable à qui on affecte cette valeur** est bien déclarée.

Cela est dû au fait que dans la méthode `getImageData` nous précisons qu'elle retourne *false* si la requête SQL ne retourne pas de contenu pour un *filename* donné.

Le tableau existe donc bien, et ses clefs également, bien que leurs valeurs soient de type null. Ainsi, lorsque nous demandons à afficher ces valeurs qui sont *null*, rien ne s'affiche mais puisque les clefs existent, aucune erreur n'est générée.

Nous testons ensuite l'ajout d'un titre et d'une description pour la nouvelle **image3.jpg** et vérifions que l'enregistrement est effectué.

Le système est ainsi mis à jour et donc fiable du point de vue de la récupération des informations et de leur restitution.

Il nous faut ensuite distinguer le cas où le contenu lié à l'image existe ou non dans la base de données. Si le contenu n'existe pas, nous utiliserons la méthode `insertImage` existante.

Si le contenu existe, nous l'affichons pour le modifier avec une requête SQL basée sur un UPDATE, et créons pour cela une méthode que nous appelons *updateImageData* (mise à jour des informations de la base de données relatives à une image).

Le code de la méthode *updateImageData* ne pose pas de difficultés particulières :

```
public function updateImageData ($title, $descr, $filename)
{
    $mysqli = new mysqli('localhost', 'root', '', 'projet_image');

    /* Vérification de la connexion */
    if ($mysqli->connect_errno) {
        echo 'Echec de la connexion ' . $mysqli->connect_error ;
        exit();
    }

    if (!$mysqli->query('UPDATE image SET title = "'. $title .'",
description = "'. $descr .'" WHERE filename = "'. $filename .'"'))
    {
        echo 'Une erreur est survenue lors de la mise à jour des données
dans la base. Message d\'erreur : ' . $mysqli->error;
        return false;
    }
    else
        return true;

    $mysqli->close();
}
```

Fig. 11

Comme d'habitude, nous prenons soin de bien vérifier la concaténation et les quotes.

Nous préparons ensuite le fichier **admin.php** pour ajouter dans le formulaire HTML un champ de type *hidden* nommé *update* lorsque le contenu des images doit être modifié et non pas inséré. La présence ou non de ce champ *update* permettra d'appeler soit la méthode *insertImage* soit la méthode *updateImageData*.

Nous vérifions si la valeur *\$image['title']* est vide ou non. Si elle vide, aucune information n'est présente dans la table pour l'image en question. Si elle n'est pas vide, l'image possède des informations dans la table et donc nous ajoutons le champ *update* afin de procéder à une mise à jour et non à une insertion.

```
<?php if(!empty($image['title'])): ?>
<input type="hidden" name="update" value="1" />
<?php endif ?>
```

Fig. 12

Pour terminer, nous modifions le fichier **process_image.php** afin de tenir compte de cet indicateur :

```
if(isset($_POST['update']))
{
    $insertImage = $image->updateImageData($title, $descr, $filename);
}
else
{
    $insertImage = $image->insertImage($title, $descr, $filename);
}
```

Fig. 13

Si le champ *update* n'est pas posté, alors `$_POST['update']` n'existe pas et donc nous insérons le titre et la description de l'image avec la méthode *insertImage*.

Si le champ *update* n'est pas posté, alors `$_POST['update']` n'existe pas et donc nous mettons à jour le titre et la description de l'image avec la méthode *updateImageData*.

B. Finalisation : ajout des informations sur la page de contenu *front*

Après avoir vérifié le bon fonctionnement de notre application suite à cette série de modifications, nous pouvons maintenant intégrer le contenu de la base dans la page contenue du front-office. L'intervention est simple puisqu'il s'agit au final de reporter les mêmes modifications que dans le fichier **admin.php** en créant les éléments HTML nécessaires pour afficher le titre et la description de chaque image s'ils existent dans la base de données.

Nous nous bornerons dans cet exemple à ajouter l'élément `<p>` pour les deux informations titre et description.

```
$images = $image->getImages(IMAGE_DIR_PATH);
?>
<h1><?php echo WEB_TITLE ?></h1>
<ul>
<?php foreach ($images as $image): ?>
<li>
<p><p>
<p><?php echo $image['title'] ?></p>
<p><?php echo $image['description'] ?></p>
</li>
<?php endforeach ?>
</ul>
```

Fig. 14

Il est important de considérer que la simplicité et la rapidité de cette modification sont rendues possibles grâce à l'utilisation des méthodes de la classe *Image*. **Le code peut être utilisé deux fois** (dans l'administration et dans les pages *front*) ce qui constitue à la fois un appréciable gain de temps et une meilleure gestion, ce qui est un des avantages de la POO.

II. La gestion des erreurs

L'architecture actuelle de l'administration est caractérisée par le fait que le fichier **process_image.php** ne fait pas l'objet d'une inclusion. Cela permet une gestion des erreurs certes fonctionnelle mais non optimisée.

Ainsi, si une erreur SQL se produit, par exemple une erreur dans la requête concernant le nommage de table, la page **process_image.php** stoppera l'utilisateur (avec un `exit`) et indiquera le message prévu à cet effet :

```
Une erreur est survenue lors de la mise à jour des données dans la
base. Message d'erreur : Table 'projet_image.images' doesn't exist
retour
```

Fig. 15

Il est toutefois plus pratique d'intégrer directement le message dans la même page afin d'avoir immédiatement l'information et d'en tenir compte sans avoir à cliquer pour revenir vers la page de gestion. Du point de vue de l'ergonomie, il s'agit d'une bonne pratique concernant une interface de gestion.

Pour ce faire, nous avons besoin de modifier l'architecture et de procéder à une inclusion du fichier **process_image.php**. Nous interviendrons également sur les contenus et la récupération des messages d'erreur ou de succès dans les méthodes et le fichier **process_image.php**.

A. Inclusion du fichier `process_image.php` dans le fichier `admin.php`

Nous effectuons classiquement un *require* pour ce fichier.

```
require('process_image.php');
```

Fig. 16

B. Modification des méthodes de la classe *Image*

Nous modifions pour les méthodes `updateImagedata` et `insertImage` le retour en cas d'erreur. En effet jusqu'à présent nous retournions `FALSE` mais nous souhaitons que le message d'erreur affiché dans l'interface soit le plus clair possible et qu'il explique concrètement quelle est la nature du problème rencontré par MySQL, cette explication étant détaillée dans la variable `$mysqli->error`.

Nous en profitons pour rendre verbeux (détaillé) le message complet afin d'obtenir cette version :

```
Une erreur est survenue lors de la mise à jour des données dans la
base.
Le message d'erreur de MySQL est : Table 'projet_image.images'
doesn't exist
Aucune information n'a été enregistrée.
```

Fig. 17

La mention « `Table 'projet_image.images' doesn't exist` » étant ici un exemple de ce que la variable `$mysqli->error` peut retourner.

Pour ce faire, nous modifions comme suit le code de la méthode *updateImage* :

```
$msg_error = 'Une erreur est survenue lors de la mise à jour des
données dans la base. <br /> Le message d\'erreur de MySQL est : '
. $mysqli->error;
$msg_error .= '<br />Aucune information n\'a été enregistrée.';
return $msg_error;
```

Fig. 18

Nous procédons de même pour la méthode *insertImage* en adaptant légèrement le texte :

```
Une erreur est survenue lors de l'insertion des données dans la
base.
```

Fig. 19



Il n'est pas conseillé de restituer les messages d'erreur SQL à tous les utilisateurs en ligne.

Les informations étant précises, elles peuvent révéler à des hackers des failles potentielles ou les aider à comprendre la structure de la base, notamment si nous affichons les messages d'erreurs complets de MySQL).

Seuls les administrateurs ayant pour mission de corriger le code sont censés obtenir ces messages. Il sera donc nécessaire d'informer l'utilisateur sans afficher toutes les informations, comme nous le verrons ultérieurement.

Ainsi notre message est-il clair et complet et indique bien à l'utilisateur qu'aucune information n'est enregistrée dans la base. À ce propos, il est souhaitable d'éviter les points d'exclamation dans les messages d'erreur. Les messages d'erreur sont suffisamment stressants pour l'utilisateur sans qu'il ne soit besoin d'en rajouter.

La ligne `return $msg_error` permet au final de retourner le message souhaité dans **process_image.php**.

C. Modification du fichier *process_image.php*

Nous commençons par supprimer dans le fichier **process_image.php** les inclusions devenues inutiles des fichiers **config.php** et **Image.php**. Ces fichiers sont en effet déjà inclus dans **admin.php** et **process_image.php** est lui-même inclus dans **admin.php**.

Ensuite, nous affectons à la variable `$msg_error` (message d'erreur) ce qui est retourné par la méthode en cas d'erreur.

```
if(true === $insertImage) // si le retour de la méthode est true
{
    $msg_success = 'Les informations ont bien été enregistrées dans la
base de données.';
}
else
{
    $msg_error = $insertImage;
}
```

Fig. 20

Ainsi le message complet édité au préalable dans les deux méthodes sera restitué dans la page d'administration.

D. Intervention sur la table : index unique pour le champ *filename*

Pour terminer, et puisque les noms de fichiers image servent d'index unique, nous allons modifier la structure de la table *image* et définir comme index unique le champ *filename*.

L'index unique empêche les doublons dans le champ concerné. Par exemple une clé primaire (*PRIMARY KEY*) est par définition un index unique. Puisque les noms de fichiers des images servent dans notre application de clés uniques, nous empêcherons ainsi des doublons.

Pour déclarer un champ comme index unique, il faut se rendre sur la structure de la table dans *phpMyAdmin* puis :

- cliquer sur *plus* ;
- et cliquer sur Ajouter un index unique.

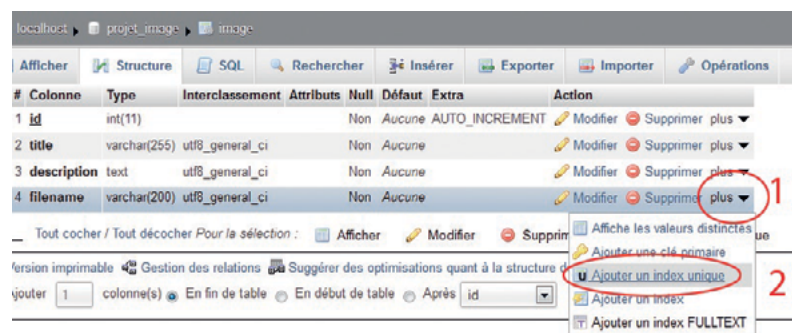


Fig.21 Ajout d'un index unique



Fig. 22

Il est également possible d'écrire la requête dans la console SQL (disponible via l'onglet SQL).

Cela est même recommandé afin de s'exercer à rédiger à la main ses propres requêtes, d'autant que la requête en question est simple :

```
ALTER TABLE image ADD UNIQUE (filename)
```

Fig.23

Nous terminons ainsi l'évolution de notre application destinée à mieux gérer les messages d'erreurs et pouvons maintenant procéder aux tests de vérification.

III. Tests de vérification, erreurs volontaires

Nous allons effectuer volontairement trois erreurs dans notre script pour tester la gestion des erreurs.

A. Première erreur volontaire

Nous allons modifier dans les deux méthodes le nom de la table *image* en *images* avec un S et tester un ajout de contenu (pour une nouvelle image) et une modification de contenus (pour une image dont le contenu est déjà enregistré).

MySQL ne reconnaîtra pas, et pour cause, la table *imageS* et va nous le signaler avec le message suivant : *Table 'projet_image.images' doesn't exist.*

B. Deuxième erreur volontaire



Nous demandons à **insérer trois valeurs** alors que **seulement deux champs** sont concernés par la requête.

Nous allons supprimer un champ dans la requête *INSERT INTO* de la méthode *InsertImage*.

```
INSERT INTO image (description, filename)
VALUES ("', $title .'", "", $descr .'", "' . $filename .'")
```

Fig.24

MySQL retournera donc le message : *Column count doesn't match value count at row 1.* (Le nombre de colonne ne correspond pas à ce qui est demandé.)

C. Troisième erreur volontaire

Nous modifions le champ *title* en *titleS* dans la requête *UPDATE* de la méthode *updateImagedata*.

MySQL retournera alors le message : *Unknown column 'titleS' in 'field list'* (la colonne *titleS* est inconnue).

Le message de MySQL sera intégré dans le message d'erreur complet. Il ne reste plus qu'à afficher le message d'erreur et à gérer une mise en forme CSS simple (un fond rouge).

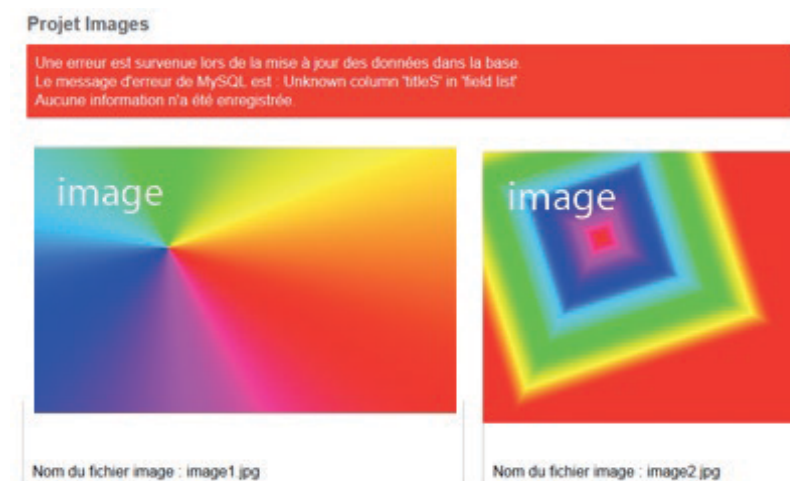


Fig.25

Une erreur est survenue lors de la mise à jour des données dans la base.
Le message d'erreur de MySQL est : Unknown column 'titleS' in 'field list'
Aucune information n'a été enregistrée.

Fig.26 Détail du message

Après avoir corrigé ces erreurs volontaires, nous testons le message de succès qui confirme le bon fonctionnement de l'application :

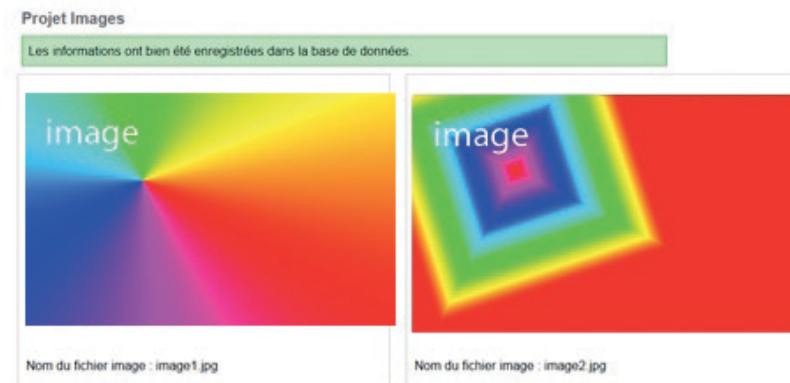


Fig.27

Le message de succès est à afficher au même endroit et bénéficie d'une mise en forme CSS simple (ici un fond vert).



Fig.28 Détail du message